W.H. Murray
C.H. Pappas

L'Assembler per l'80286/80386



L'Assembler per l'80286/80386

W.H. Murray
C.H. Pappas

L'Assembler per l'80286/80386

McGraw-Hill Libri Italia srl

Milano · New York · St. Louis · San Francisco · Oklahoma City
Auckland · Bogotá · Caracas · Hamburg · Lisboa · London · Madrid
Mexico · Montreal · New Delhi · Paris · San Juan · São Paulo
Singapore · Sydney · Tokyo · Toronto

Da un originale Osborne/McGraw-Hill

Titolo originale: *80386/80286 Assembly Language Programming*

Realizzazione editoriale: EDIGEO srl, via del Lauro 3, 20121 Milano
Traduzione: Giorgio Pieragostini
Revisione: Anna Antola e Nello Scarabottolo
Copertina: Marco Mazzucato
Composizione e stampa: Litovelox, Trento

ISBN 88-386-0049-X

1ª edizione ottobre 1987
2ª ristampa novembre 1991

Printed in Italy
34567890LTVLET9054321

Indice

Introduzione

Gli scopi che questo libro si prefigge di raggiungere sono principalmente tre: introdurre alla programmazione in linguaggio assemblatore, insegnare – tramite esempi di difficoltà crescente – le tecniche di programmazione in linguaggio assemblatore sui microprocessori 80386/80286 e sui coprocessori 80387/80287 e fornire un testo di consultazione per conoscere – e utilizzare correttamente – le istruzioni del linguaggio assemblatore e le soluzioni programmative più adatte al tipo di problema da risolvere.
Il linguaggio assemblatore 80386/80286 rappresenta l'evoluzione del linguaggio assemblatore 8088/8086, per cui, pur mantenendo con quest'ultimo completa compatibilità (molti programmi, infatti, che vengono presentati in questo libro, possono essere eseguiti anche su calcolatori dotati di microprocessore 8088/8086), introduce concetti innovativi, come ad esempio la programmazione in "modalità protetta".
Il linguaggio assemblatore permette al programmatore di interagire direttamente con il calcolatore, a differenza di quanto accade con i linguaggi di alto livello – come Pascal, Ada, FORTRAN, ecc. – dove l'attività del programmatore è strettamente dipendente dal tipo di compilatore disponibile.
Al contrario, il programmatore in linguaggio assemblatore può codificare – sotto forma di routine – le istruzioni che non sono supportate dal compilatore, potenziando così il linguaggio di alto livello, oppure può utilizzare le routine BIOS che risiedono nella ROM del calcolatore.
Nel libro verranno inoltre presentati, discussi e documentati numerosi programmi (ognuno dei quali può essere codificato, tradotto ed eseguito senza problemi), in modo che sia il neofita che il programmatore esperto siano in grado di capire il significato delle singole istruzioni e lo stile di programmazione adottato.

1

Introduzione al linguaggio assemblatore

La sola conoscenza dei linguaggi di programmazione di alto livello non è sempre sufficiente per avere pieno controllo sul calcolatore: a volte è indispensabile conoscere anche l'architettura circuitale della macchina su cui si lavora, vale a dire come viene gestita la memoria, quanto rapida è l'esecuzione di un'istruzione, come viene realizzata la protezione dei dati e del codice, quali sono le periferiche a disposizione dell'utente nonché i comandi necessari al loro utilizzo, ecc.

Con il linguaggio assemblatore, il programmatore acquisisce una conoscenza completa dell'architettura del calcolatore, in quanto viene messo in condizione di accedere direttamente ai registri e alla memoria della macchina, con una visibilità che si spinge a livello di singolo bit.

Un programma scritto in linguaggio assemblatore produce, una volta assemblato, un codice eseguibile più veloce rispetto a quello ottenuto dalla compilazione o dalla interpretazione di programmi scritti in linguaggi di alto livello. Infatti, il linguaggio assemblatore si compone di un insieme di istruzioni che, poste in una particolare sequenza, indicano direttamente al microprocessore le azioni da intraprendere.

Questo vantaggio ha però un prezzo. Confrontando un programma scritto in linguaggio assemblatore con un programma scritto in linguaggio di alto livello, risulta chiara la differenza tra i due, in termini di leggibilità ed eleganza programmativa: molte operazioni che vengono realizzate con una singola istruzione nel linguaggio di alto livello, devono essere esplicitamente codificate con più istruzioni nel linguaggio assemblatore. Inoltre, un programma scritto in un linguaggio di alto livello risulta più leggibile, più facile da correggere e più portabile di un programma scritto nel linguaggio assemblatore.

Ad ogni modo, la scelta del linguaggio di programmazione da utilizzare dipende dal tipo di applicazione che si intende realizzare: se questa richiede una grande quantità di codice ed è soggetta a continue modifiche e aggiornamenti, è preferibile l'uso di un linguaggio di alto livello, a meno che non sia indispensabile un codice ad elevata velocità di esecuzione, estremamente efficiente, nel qual caso è consigliabile l'uso del linguaggio assemblatore.
Molti concetti, che risultano fondamentali nella programmazione in linguaggio assemblatore, sono completamente sconosciuti a chi programma in un linguaggio di alto livello; con il linguaggio assemblatore, infatti, il programmatore deve spesso scegliere la posizione fisica e la dimensione delle aree di memoria in cui allocare il codice e i dati; deve definire, per ogni dato, il tipo (stringa, BCD, statico, dinamico, con segno, senza segno, reale, in virgola mobile, ecc.) e la dimensione (byte, word, doubleword, quadword, tenbyte); deve verificare se il calcolatore dispone di coprocessori matematici, nel qual caso decidere se e come convertire il formato delle variabili, per adattarlo a quello richiesto dai coprocessori; deve infine sapere quali registri può utilizzare nelle operazioni di trasferimento di dati e con quali porte può accedere all'interfaccia video a colori, all'interfaccia video monocromatico o alla stampante.

1.1 Vantaggi principali: velocità e controllo delle operazioni

Uno dei principali vantaggi offerti dalla programmazione in linguaggio assemblatore è la velocità con cui il codice viene eseguito.
Infatti, per eseguire una qualunque istruzione, scritta dal programmatore in forma mnemonica e tradotta da un programma assembler nella codifica binaria (o linguaggio macchina) del microprocessore, occorrono pochi microsecondi.
Un programma interprete, invece, esamina il codice sorgente di alto livello e lo traduce linea per linea, sostituendo le istruzioni con procedure predefinite che fanno parte della sua libreria e che sono già codificate nel linguaggio macchina. Poiché ogni esecuzione del programma sorgente richiede una nuova valutazione delle espressioni e nuovi controlli sintattici e semantici, anche se il codice rimane invariato e cambiano solo i dati di ingresso, un codice interpretato ha una velocità di esecuzione molto ridotta.
Un programma compilatore, infine, esamina il codice sorgente e lo traduce direttamente in una sequenza di istruzioni macchina che il microprocessore è in grado di eseguire. I programmi compilati hanno una velocità di esecuzione maggiore rispetto a quella dei programmi interpretati, in quanto la traduzione viene eseguita una sola volta, indipendentemente dalla modifica dei dati di ingresso. La compilazione di un programma presenta però uno

svantaggio: i compilatori devono infatti codificare una grande varietà di funzioni e generano, di conseguenza, un codice di elevate dimensioni anche per programmi sorgenti ridotti, per cui un programma compilato richiede una quantità di memoria maggiore rispetto a quella richiesta dallo stesso programma scritto nel linguaggio assemblatore.
Non si deve dimenticare, inoltre, che il linguaggio assemblatore permette al programmatore di interagire con il sistema operativo, di scrivere un insieme di procedure che realizzano nuove funzioni di sistema, in aggiunta a quelli disponibili, e di controllare direttamente le operazioni di ingresso e uscita su video, stampante e unità disco.

1.2 La famiglia 80286/80386

Lo sviluppo dei microprocessori è in rapida espansione e ogni anno vengono realizzati nuovi prodotti, sempre più potenti e affidabili, che spesso sostituiscono i precedenti.
Il primo circuito integrato è stato costruito all'inizio degli anni Sessanta e ha rappresentato una svolta fondamentale nell'evoluzione dell'elettronica; da quel momento in poi, infatti, è divenuto possibile integrare condensatori, diodi e transistori su un'unica piastrina di silicio (chip) di dimensioni estremamente ridotte, a vantaggio di una complessità e compattezza dei circuiti sempre maggiori.
Dieci anni più tardi, la Intel ha sviluppato il primo microprocessore a 8 bit su singolo chip, l'8008, seguito, nel 1974, dall'8080 che può considerarsi il capostipite della seconda generazione di microprocessori. Poco tempo dopo anche la Zilog ha messo in commercio un microprocessore a 8 bit, lo Z-80, dando inizio a una concorrenza sfrenata tra le case costruttrici di microprocessori.
Solo quattro anni più tardi nasce la cosiddetta terza generazione di microprocessori, con lo sviluppo, da parte della Intel, del microprocessore a 16 bit 8086. Anche se questo chip costituisce un'evoluzione dell'8080, molte delle sue caratteristiche funzionali risultano innovative. La Intel, come variante all'8086, sviluppa anche l'8088 (la versione a 8 bit dell'8086), che rimane uno dei microprocessori più avanzati per quell'epoca. L'architettura dell'8088, in particolare, offre prestazioni di poco inferiori rispetto a quelle dell'8086 e, nello stesso tempo, è compatibile con le unità periferiche disponibili in quel momento (tutte a 8 bit). Per questi motivi, la IBM decise di utilizzarlo su tutti i suoi personal computer.
Quasi contemporaneamente, la Intel rende disponibile il coprocessore matematico 8087, cioè un elaboratore numerico ad alta velocità, adatto ad elaborazioni matematiche ad elevata precisione, che opera in parallelo con l'8088/8086.

Ma la Intel non si ferma qui. Nel 1984 sviluppa l'80286, un microprocessore che costituisce il precursore dell'attuale generazione di CPU (Central Processing Unit: unità centrale di elaborazione) adatte a supportare applicazioni nelle quali siano richieste prestazioni molto elevate. Oltre alla compatibilità con l'8088/8086 che gli permette di eseguire tutto il software sviluppato per i precedenti microprocessori, l'80286 integra sullo stesso chip VLSI (Very Large Scale Integration: densità di integrazione molto elevata) diverse funzionalità innovative, come ad esempio la gestione della memoria virtuale e la protezione hardware tra processi concorrenti.
L'80286 risulta compatibile con l'8088/8086, in quanto mantiene, potenziandoli, il set di istruzioni e i modi di indirizzamento, e supporta linguaggi di alto livello, come il Pascal, il PL/M e il C, grazie all'inserimento nella CPU di registri di lavoro particolarmente adatti all'esecuzione di codice generato da un compilatore.
L'80286 supporta diversi tipi di strutture dati, come ad esempio stringhe di caratteri, numeri nei formati BCD o in virgola mobile, e consente anche efficienti modalità di indirizzamento a strutture dati più complesse, quali array statici o dinamici, record e array di record.
La struttura della memoria dell'80286 ben si adatta alle tecniche di programmazione modulare, in base alle quali un programma viene suddiviso in segmenti di dimensioni variabili che costituiscono le unità di memoria trattate dal microprocessore. La segmentazione della memoria permette di realizzare un codice più compatto (poiché i riferimenti – o indirizzi – all'interno di un segmento possono avere dimensioni inferiori), più semplice da correggere e aggiornare, e garantisce una sofisticata gestione della memoria stessa, attraverso meccanismi di protezione e di indirizzamento virtuale.
Lo spazio di memoria offerto dall'80286 è molto esteso e quindi adatto a soddisfare le esigenze delle attuali applicazioni: la memoria reale ha, infatti, una dimensione di 16 megabyte (2^{24} byte), suddivisa tra RAM e ROM, ed è quindi in grado di contenere programmi molto ampi e strutture dati complesse, minimizzando i tempi di accesso ad esse.
Per le applicazioni che richiedono una gestione dinamica della memoria, l'80286 opera anche in modalità virtuale, rendendo disponibile ad ogni utente uno spazio di indirizzamento logico di 1 gigabyte (2^{30} byte).
L'architettura dell'80286 supporta, in modo molto efficiente, un ambiente di lavoro multiutente e concorrente, tipico di applicazioni di gestione di attività in tempo reale. Le elevate prestazioni raggiungibili con il microprocessore 80286 hanno permesso di triplicare il numero di utenti che utilizzano il sistema e di ottenere tempi di risposta anche sei volte più rapidi.
L'ultimo nato della famiglia Intel è, al momento, il microprocessore 80386: si tratta di un componente a 32 bit – cioè la dimensione del suo bus dati esterno è di 32 bit, il doppio rispetto a quella dell'80286 – che può indirizzare 4 gigabyte di memoria.

1.3 Scopi del libro

Lo scopo principale di questo libro è quello di insegnare la programmazione nel linguaggio assemblatore 80286/80386, indicando per ogni istruzione del linguaggio il significato e le modalità di utilizzo. In questo modo il lettore prende coscienza di come il calcolatore lavora a livello di registri e di memoria, di quali vantaggi e svantaggi presenta la programmazione nel linguaggio assemblatore e, a seconda del tipo di applicazione che intende realizzare, di quale versione di programma assemblatore conviene servirsi (IBM, Microsoft o Speedware), fino a raggiungere una conoscenza completa dell'architettura interna dell'80286/80386 e dell'80287/80387.
Il testo insegna anche il significato e l'uso delle direttive all'assembler, delle macro, delle procedure e spiega come creare e manipolare una libreria di macro.
I numerosi esempi di programmi presenti nel libro indicano come utilizzare le procedure di gestione dei file e di accesso alle interfacce grafiche presenti nel calcolatore, sfruttando l'estrema velocità e precisione dei coprocessori matematici 80287 e 80387. Inoltre, il testo spiega dettagliatamente come interfacciare i programmi di alto livello con i programmi scritti nel linguaggio assemblatore, come richiedere i servizi del sistema operativo, attraverso le procedure DOS e BIOS, e come utilizzare il sistema di ricerca e correzione degli errori (programma Debug).

1.4 Conoscenze necessarie per la lettura del testo

Il lettore deve conoscere almeno un linguaggio di alto livello – come il C, il Pascal o il BASIC – in quanto il testo propone numerosi riferimenti ad esso, per spiegare le caratteristiche del linguaggio assemblatore.
È utile, ma non necessaria, una certa familiarità con il sistema di numerazione esadecimale e la conoscenza delle operazioni logiche AND, NAND, OR, NOR, XOR, SHIFT LEFT/RIGHT, ROTATE, COMPLEMENT, ecc.

1.5 Sistemi di numerazione

La conoscenza della modalità di memorizzazione delle informazioni nel calcolatore costituisce un requisito indispensabile per programmare in linguaggio assemblatore.
Il calcolatore è una macchina elettrica che capisce e interpreta solo informazioni rappresentate sotto forma di tensione. In particolare, il microprocessore basa il suo funzionamento sul riconoscimento di due livelli di tensione

(alto/basso) che equivalgono, in termini logici, ai valori 1/0. Questa coppia di valori rappresenta la più piccola informazione trattata dal calcolatore e viene indicata comunemente con il termine *bit*.
L'uomo utilizza, per i suoi calcoli, il sistema di numerazione decimale, mentre il calcolatore, come abbiamo detto, distingue solo i due valori 0 e 1. Esiste quindi un'incompatibilità di linguaggio tra l'uomo e la macchina che ha portato allo sviluppo di numerosi sistemi per la codifica delle informazioni, il più importante dei quali è il sistema ASCII (American Standard Code for Information Interchange), con cui è possibile rappresentare tutti i simboli della tastiera di una macchina da scrivere. Contemporaneamente sono state sviluppate diverse codifiche numeriche con lo scopo di rappresentare i numeri in un formato che fosse compatibile con il calcolatore, come ad esempio la codifica in complemento a due, che permette di eseguire facilmente operazioni di somma e sottrazione su numeri positivi e negativi.
Una delle più importanti codifiche numeriche, per chi intende programmare nel linguaggio assemblatore, è costituita dalla codifica esadecimale, che permette un sensibile miglioramento della leggibilità delle istruzioni, degli indirizzi di memoria e dei dati, altrimenti espressi nel formato binario.

NUMERI BINARI

Il sistema di numerazione decimale deriva il suo nome dal numero di simboli – dieci – che costituiscono gli elementi di codifica (simboli – o cifre – da 0 a 9); il sistema di numerazione binario, invece, utilizza due soli simboli: 0 e 1.
Consideriamo ad esempio il numero 1024, definito secondo la rappresentazione posizionale, da quattro unità, due decine, zero centinaia e una migliaia (procedendo da destra verso sinistra si ha un incremento del peso relativo alla cifra considerata). Questo numero decimale – cioè rappresentato in base 10 – può quindi essere scritto nella forma:

1 0 2 4

$4 * 10^0 = 4$ Meno significativo
$2 * 10^1 = 20$
$0 * 10^2 = 000$
$1 * 10^3 = 1000$ Più significativo

1024

È possibile ottenere il numero moltiplicando ogni sua cifra per la base del sistema di numerazione adottato, elevata ad una potenza pari al peso (posizione) che la cifra stessa possiede (occupa) all'interno del numero. Le potenze si incrementano di una unità muovendoci dalla posizione meno significativa alla posizione più significativa.

Questo semplice principio di conversione può essere applicato anche a qualunque numero binario. Esaminiamo, infatti, come il numero binario 1010 viene convertito nel corrispondente numero decimale:

```
1 0 1 0
| | | └────────────► 0 * 2^0  =  0  Bit meno significativo
| | └──────────────► 1 * 2^1  =  2
| └────────────────► 0 * 2^2  =  0
└──────────────────► 1 * 2^3  =  8  Bit più significativo
                                ──
                                10
```

In questo caso, ad ogni cifra binaria 1 si sostituisce il valore numerico decimale ottenuto elevando la base del sistema binario – cioè 2 – ad una potenza coincidente con la posizione che la cifra occupa all'interno del numero, partendo dal bit meno significativo (Least Significant Bit: LSB) e procedendo verso il bit più significativo (Most Significant Bit: MSB). Lo stesso procedimento rimane valido se si vuole convertire in decimale un numero espresso in una qualunque altra base di numerazione: è sufficiente sostituire alla base 2 la base prescelta e ripetere i passi precedenti.

La conversione, invece, da decimale a binario viene realizzata semplicemente sottraendo al numero decimale la massima potenza di due in esso contenuta e ripetendo la stessa operazione sul risultato della sottrazione, fino a quando non si giunge a considerare 2^0 (cioè 1). A questo punto l'operazione di conversione si arresta e il numero binario risultante viene ottenuto disponendo in sequenza i coefficienti delle potenze di due sottratte in successione dal numero decimale. Consideriamo ad esempio il numero decimale 11.

Al numero 11 si sottrae la massima potenza di due in esso contenuta, cioè $1 \cdot 2^3 = 8$ (quindi questa sottrazione equivale ad avere 1 come bit più significativo del numero binario): il risultato è 3. Poiché $2^2 = 4$ è maggiore di 3, la sottrazione successiva non è possibile e quindi si aggiunge 0 al numero binario precedente, ottenendo 10. A 3 è possibile sottrarre solamente $2^1 = 2$ (questa sottrazione aggiunge un bit 1: 101), ottenendo come risultato 1, cui può essere sottratto il valore $2^0 = 1$ (questa sottrazione aggiunge un bit 1: 1011).

Un metodo di conversione più semplice e adatto a trattare numeri binari di maggiori dimensioni consiste nell'eseguire divisioni ripetute, utilizzando come divisore la base del sistema di numerazione in cui si intende convertire il numero e come dividendo il numero da convertire. Ad esempio, la conversione del numero 50, nel sistema di numerazione binaria, procede nel modo seguente:

Dividendo	Divisore	Risultato	Resto	
50	2	25	0	► Risposta = 110010
25	2	12	1	
12	2	6	0	
6	2	3	0	
3	2	1	1	
1	2	0	1	

È possibile utilizzare questo metodo per convertire qualunque numero decimale in una qualunque altra base; la base prescelta diventa il nuovo divisore.

SOMMA E SOTTRAZIONE BINARIA

Le operazioni binarie di somma e sottrazione sono identiche alle analoghe operazioni decimali, ad eccezione del riporto generato e del prestito richiesto, che interessano, nel primo caso, una potenza di 2, mentre nel secondo caso una potenza di 10. Consideriamo i seguenti due esempi:

25 +85 ——— 110	In questo esempio, sommando le due cifre 5, si genera un riporto (1) che viene aggiunto alla colonna adiacente più significativa, cioè quella delle decine. A questo punto, sommando 8, 2 e il riporto, si genera un altro riporto (1) per la colonna delle centinaia. Si ottiene alla fine il risultato 110.

Vediamo ora l'addizione binaria:

1010 (10) +0011 (3) ——— 1101 (13)	In questo esempio, invece, la somma di 0 e 1 genera come risultato 1, senza alcun riporto. Sommando 1 e 1, si ottiene 2, che in binario si esprime con 10. Il bit 1 della sequenza 10 costituisce il riporto alla posizione adiacente (più significativa). Si ottiene alla fine il risultato 1101.

Il prossimo esempio illustra l'unica situazione non ancora incontrata nella somma di due numeri binari:

0011 (3) +0011 (3) ——— 0110 (6)	I primi due 1 generano un riporto da sommare alla seconda colonna che si compone già di due 1, per cui gli 1 da sommare diventano tre. Esaminiamo più dettagliatamente i passi da compiere.
1 1 +1 ——— 11	La somma di due 1 in seconda colonna dà come risultato 10 e, sommando ad esso 1, che è il riporto della prima colonna, si ottiene, in binario, la sequenza 11, di cui il bit più significativo costituisce il riporto per la terza colonna, mentre il bit meno significativo rappresenta il risultato della somma in seconda colonna.

L'operazione di sottrazione binaria utilizza le stesse regole adottate nella sottrazione decimale. L'unica differenza è rappresentata dal prestito richiesto che, nel primo caso, è una potenza di 2, mentre, nel secondo caso, è una potenza di 10.

534 −251 ——— 283	Non c'è difficoltà a sottrarre la cifra decimale 1 alla cifra decimale 4. Quando, invece, si cerca di sottrarre 5 al valore 3, è necessario richiedere un prestito alla cifra adiacente più significativa ($1 \cdot 10^2$) e sottrarre così 50 a 130. Proseguendo, si sottrae senza difficoltà 2 a 4 (5 meno il prestito precedente), per cui il risultato finale è 283.

```
 1011
-0110
-----
 0101
```

Nel caso di sottrazione di due numeri binari, non c'è difficoltà a sottrarre 0 a 1 (prima colonna), così come a sottrarre 1 a 1 (seconda colonna). Esiste, invece, qualche problema a sottrarre 1 a 0 (terza colonna). Questa operazione richiede un prestito dal bit adiacente più significativo in quarta colonna, il cui peso è $1 \cdot 2^3$. Sottraiamo, quindi, $1 \cdot 2^2 = 4$ a $1 \cdot 2^3 = 8$, ottenendo $1 \cdot 2^2 = 4$, che fornisce il bit 1 da collocare in terza colonna. Il risultato finale è dunque 0101.

BYTE

Prima di continuare l'esame dei diversi sistemi di numerazione, è bene ricordare che il codice e i dati di un programma vengono memorizzati e, successivamente, utilizzati da un calcolatore la cui architettura interna stabilisce il loro formato e il campo dei valori che essi possono assumere.
I calcolatori non memorizzano casualmente numeri binari di lunghezza qualsiasi, ma il campo di valori manipolabili dipende dalla dimensione della cella di memoria in cui il dato viene allocato. Nel microprocessore 80286, ad esempio, la dimensione di una singola locazione di memoria risulta di 8 bit (1 byte) e le posizioni dei singoli bit sono numerate da 0 (LSB o Least Significant Bit) a 7 (MSB o Most Significant Bit).
La Figura 1.1 mostra alcune locazioni di memoria consecutive, ciascuna delle quali contiene un byte di informazione che può essere costituito da una istru-

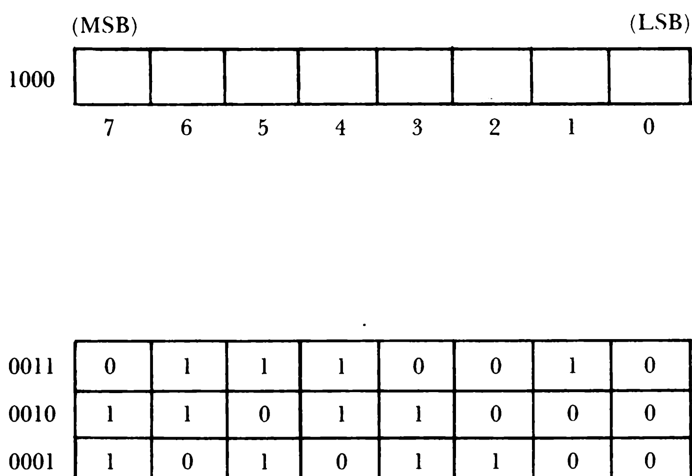


Figura 1.1 Esempio di locazioni di memoria consecutive

zione macchina, da un indirizzo ad un'altra locazione di memoria, da un carattere o da un numero.
Otto bit possono generare fino a 256 valori distinti, cosicché una locazione di memoria è in grado di contenere la rappresentazione binaria di tutti i numeri positivi compresi tra 0 e 255, oppure un carattere ASCII (per avere maggiori informazioni sulla codifica ASCII, si consulti l'Appendice E).
Il termine tecnico generalmente utilizzato per indicare il numero di bit memorizzati in una locazione di memoria è "parola". I primi calcolatori avevano una architettura interna di 4 bit, per cui una parola indicava quattro bit. Alcuni calcolatori dispongono di locazioni di memoria di 64 bit, per cui una parola è costituita, in questo caso, da una stringa di 64 bit. L'architettura dell'80286 interpreta una parola come una sequenza di 16 bit, cioè l'insieme di due byte, ognuno dei quali può essere diviso in due gruppi di 4 bit (4 bit costituiscono un nibble), che possono memorizzare, ciascuno, una cifra esadecimale.

CARATTERI

Gli otto bit che costituiscono un byte possono rappresentare anche valori non numerici. Infatti, con la codifica ASCII a 7 bit è possibile rappresentare simbolicamente tutti i caratteri alfabetici e numerici (oltre a speciali codici di controllo), assegnando una arbitraria codifica binaria ad ogni lettera, cifra e carattere speciale, presente ad esempio sulla tastiera di qualunque macchina da scrivere.
Una codifica di 7 bit è in grado di rappresentare 128 simboli distinti; l'ottavo bit viene di solito utilizzato come bit di parità per scoprire eventuali errori di trasmissione. Alcuni costruttori di generatori di caratteri a ROM utilizzano l'ottavo bit per estendere il set di caratteri. Aggiungendo, infatti, un altro bit ai sette già disponibili, il numero di simboli rappresentabili raddoppia (da 128 a 256), per cui è possibile codificare anche simboli matematici e, soprattutto, simboli grafici.

NUMERI CON SEGNO

Esaminiamo ora come viene rappresentato, nell'architettura dell'80286, un numero intero. Se tutti gli otto bit concorrono alla memorizzazione di un numero positivo, è possibile rappresentare valori compresi tra 0 (00000000) e 255 (11111111). Sommando 1 a 255 (11111111) si supera la capacità di memorizzazione del microprocessore: si otterrebbe infatti 100000000. Poiché questo numero non può essere rappresentato, viene operata automaticamente una operazione di "troncamento" che elimina il primo bit, cioè l'1, e si ritorna al valore iniziale (00000000).

La rappresentazione dei numeri sia positivi che negativi (numeri relativi) riduce il campo dei valori memorizzabili, in quanto è necessario utilizzare un bit, degli otto a disposizione, per indicare il segno del numero. In questo modo, rimangono 7 bit per rappresentare il valore numerico assoluto (o modulo), che risulta così compreso tra 0 (0000000) e 127 (1111111), e quindi si verifica un dimezzamento del campo di valori assoluti che si possono rappresentare con otto bit.
Nella rappresentazione dei dati in modulo e segno (sign and magnitude) il segno del numero viene memorizzato nel bit più significativo: se questo assume il valore 0, il numero è positivo, mentre se assume il valore 1, il numero è negativo.

```
0000 0101   rappresenta il numero +5
1000 0101   rappresenta il numero -5
```

Nel caso di memorizzazione del numero 0, teoricamente si ha che:

```
0000 0000   rappresenta il numero +0
1000 0000   rappresenta il numero -0
```

Avendo definito una nuova rappresentazione dei numeri, diventa indispensabile definire un nuovo insieme di regole aritmetiche. La prima assunzione da fare è di rappresentare sempre il numero 0 come valore positivo, cioè con la sequenza di bit 0000 0000.
Se si utilizza l'ottavo bit come indicatore di segno, i restanti sette bit consentono di rappresentare i numeri compresi tra −127 e +127.
Consideriamo ora le operazioni aritmetiche:

```
  0000 0000    +0
 -0000 0001    +1
 ---------------
  1111 1111  -127
```

Questo esempio non genera un risultato corretto. Per poter eseguire nel modo usuale le operazioni aritmetiche su numeri con segno, il numero −1 dovrebbe essere rappresentato dalla sequenza di bit 1111 1111.
Nel prossimo paragrafo vedremo come risolvere questo problema.

COMPLEMENTO A DUE

Perché le operazioni di somma e di sottrazione operino correttamente sui numeri dotati di segno, è necessario utilizzare la notazione in complemento a due. Rappresentando, infatti, i numeri negativi a mezzo dei complementi, l'operazione di sottrazione viene ricondotta ad una operazione di somma: (X − Y) diventa (X + (− Y)). La rappresentazione in complemento a due dei numeri positivi rimane identica alla rappresentazione in modulo e segno esaminata nel paragrafo precedente, mentre per ottenere la rappresentazione in complemento a due dei numeri negativi si considera il valore positivo del numero binario e si complementa ogni suo bit, sommando 1 al risultato ottenuto.
La notazione in complemento a due dei numeri positivi e negativi risulta dunque la seguente:

0000 0100	+4
0000 0011	+3
0000 0010	+2
0000 0001	+1
0000 0000	0
1111 1111	−1
1111 1110	−2
1111 1101	−3
1111 1100	−4

Anche nel caso della rappresentazione in complemento a due il bit più significativo fornisce una indicazione del segno del numero e il risultato delle operazioni di somma e di sottrazione in complemento a due deve essere anch'esso un numero espresso nella notazione in complemento a due. Ad esempio:

```
   0000 0100    +4
  +1111 1101    −3
 ---------------
 1) 0000 0001   +1   (il riporto generato viene ignorato)
```

Il seguente esempio indica come si deve procedere per ricavare dal numero +5 la rappresentazione in complemento a due di −5:

```
  0000 0101    +5
  1111 1010    (ogni bit viene complementato e viene aggiunto +1)
 +0000 0001
 ----------
  1111 1011    −5
```

Per ricavare dalla rappresentazione in complemento a due di −4 la rappresentazione in complemento a due di +4, si segue lo stesso procedimento:

```
 1111 1100    -4
 0000 0011    (ogni bit viene complementato e viene aggiunto +1)
+0000 0001
----------
 0000 0100    +4
```

Consideriamo un altro esempio:

```
   1111 1001   -7  (tutti i numeri sono in complemento a due)
  +1111 1000   -8
  ----------
1) 1111 0001  -15
```

1.6 Estensione del bit di segno

Poiché i registri interni dell'80286 sono di 16 bit e la dimensione di una locazione di memoria è di un byte, è possibile che, ad un certo punto del programma, sia necessario sommare due numeri memorizzati tramite un numero di bit differente. Ad esempio, la somma di un numero di 8 bit in complemento a due con un numero di 16 bit, espresso nella stessa notazione, viene effettuata nel modo seguente:

```
           0000 0000   +5
+1111 1111 1111 1101   -3
-----------------------
```

In questo esempio, è necessario far precedere al bit più significativo di +5 la sequenza 0000 0000 (estensione del segno), mentre se il numero di 8 bit è negativo, come ad esempio:

```
           1111 1111   -1
+0000 0000 0000 0100   +4
-----------------------
```

è necessario far precedere la sequenza 1111 1111 (estensione del segno). La regola generale sull'estensione del bit di segno dice che − a seconda del segno del numero − si deve aggiungere la sequenza di bit 0 (positivo) o 1 (negativo) al bit più significativo del numero.

```
0000 0000 0000 0101  (rappresentazione a 16 bit di +5)
1111 1111 1111 1111  (rappresentazione a 16 bit di -1)
```

NUMERI ESADECIMALI

Il sistema di numerazione esadecimale deriva il suo nome dal numero di simboli di codifica utilizzati: 16. La notazione esadecimale è particolarmente adatta, nel linguaggio assemblatore, alla scrittura e alla lettura di codice, di dati o di etichette di locazioni di memoria, in quanto ogni simbolo esadecimale può essere rappresentato da un gruppo di 4 bit.
I numeri in notazione esadecimale contengono le cifre comprese tra 0 e 9 e le lettere A, B, C, D, E e F. La Tabella 1.1 mostra la relazione esistente tra le codifiche decimale, binaria ed esadecimale.
Il prossimo breve esempio illustra il motivo che ha reso così utile la notazione numerica esadecimale. Rappresentiamo il contenuto di una sequenza di locazioni di memoria consecutive nel formato binario:

```
10010110
11100011
10101011
00101100
```

e nel formato esadecimale:

```
96
E3
AB
2C
```

Tabella 1.1 Rappresentazioni numeriche decimale, binaria ed esadecimale

Decimale	Binario	Esadecimale
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

10100110

Passo 1. Divisione del byte in due nibble.

1010 0110

Passo 2. Conversione di ogni nibble nel formato decimale.

10 6

Passo 3. Sostituzione con la codifica esadecimale.

A 6

Figura 1.2 Conversione da binario a esadecimale

Indubbiamente la seconda sequenza di caratteri è più compatta, semplice da scandire ed, eventualmente, da ricordare.
La conversione esadecimale di un numero binario viene realizzata in modo molto semplice: partendo dal bit meno significativo e procedendo verso sinistra, si seziona il numero binario in gruppi di 4 bit, si sostituisce ad ogni nibble così ottenuto il corrispondente valore decimale che, a sua volta, viene convertito in esadecimale. La Figura 1.2 illustra questo procedimento.
L'operazione di conversione da esadecimale a decimale utilizza gli stessi principi seguiti per realizzare la conversione da binario a decimale.
La Figura 1.3 mostra come eseguire la conversione del numero esadecimale FACE nel formato decimale. La rappresentazione binaria di questo numero (64206) è la seguente:

1111101011001110

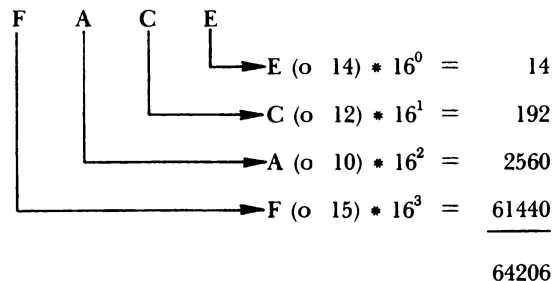


Figura 1.3 Conversione da esadecimale a binario

Il numero, debitamente sezionato:

11111010 11001110

viene memorizzato in due locazioni di memoria e la sua rappresentazione esadecimale è quindi FA CE.

1.7 Gruppi di bit di dimensioni standard

L'architettura dell'80286 e dell'80386 riconosce e manipola, come singole unità di informazione, sequenze di bit aventi una dimensione maggiore di 8.

WORD

Poiché l'80286 è un microprocessore a 16 bit che è stato quindi progettato per poter lavorare su valori espressi in 16 bit, è conveniente unire due byte in una word (parola), come è indicato nella Figura 1.4. Una word aumenta la capacità di memorizzazione da 255 (FF in esadecimale) a 65535 (FFFF in esadecimale). Il programma assemblatore memorizza e manipola gli interi

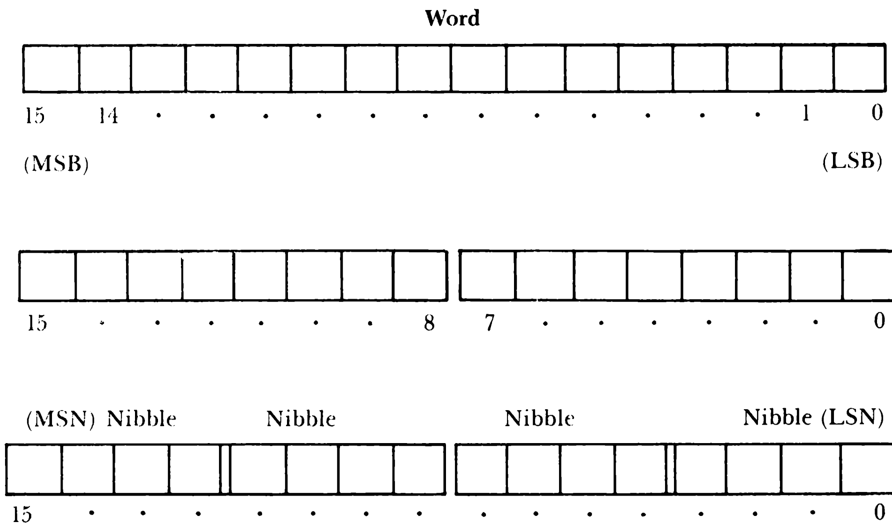


Figura 1.4 Suddivisione di una word in bit, nibble e byte

nel formato di word. Per indirizzare le locazioni di memoria in cui sono contenuti i dati di tipo stringa e le variabili di tipo intero, si utilizzano puntatori a 16 bit, che assumono valori compresi tra 0000H e FFFFH. È possibile così indirizzare fino a 64 kB di memoria. 1 chilobyte è "circa" 1000 byte; per le unità di misura in byte si usa infatti il moltiplicatore $2^{10}=1024$ invece di $10^3=1000$. 64 kB sono quindi 64 x 1024=65 536 byte.
L'ordine con cui vengono allocati in memoria i due byte di un numero intero di 16 bit è inverso rispetto all'ordine con cui vengono manipolati nel programma: il byte meno significativo infatti viene memorizzato per primo, mentre il byte più significativo viene memorizzato nella posizione seguente, verso valori di indirizzi crescenti.
Ad esempio, l'allocazione in memoria del valore 3456H procede nel modo seguente:

Indirizzo di memoria	Valore
0000 0001	34H
0000 0000	56H

Il programmatore non deve assolutamente preoccuparsi di come vengono memorizzati i 16 bit di una word: ogni istruzione, che referenzia un dato in memoria, ne interpreta correttamente il formato di memorizzazione ed esegue senza errori tutti gli eventuali trasferimenti.

DOUBLEWORD

Una doubleword (doppia parola) è costituita dall'unione di due word (allocate in posizione consecutiva in memoria) e presenta quindi una dimensione di 32 bit, come viene illustrato nella Figura 1.5.

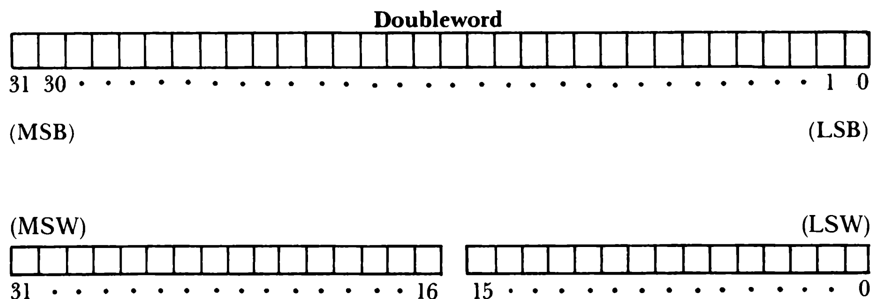


Figura 1.5 Byte e word allocate in una doubleword

Questo formato è particolarmente utile nell'indirizzamento della memoria dell'80286, in quanto un indirizzo a 32 bit permette un accesso diretto a ben più di un milione di byte. Inoltre, il formato doubleword garantisce una maggiore precisione nelle operazioni aritmetiche, per via della maggiore ampiezza dell'intervallo di valori utilizzabile. Un dato a 32 bit può rappresentare sia numeri molto grandi che numeri molto piccoli, sia in notazione intera che in virgola mobile. La tecnica di allocazione in memoria di una doubleword è la stessa di quella seguita per una word: i 16 bit meno significativi vengono memorizzati a partire dalla locazione di memoria di indirizzo inferiore, mentre i 16 bit più significativi vengono allocati, in posizione consecutiva, verso gli indirizzi di memoria crescenti. Ogni gruppo di 16 bit è naturalmente diviso in due byte e la doubleword viene dunque allocata in memoria partendo dal byte meno significativo fino al byte più significativo. Ad esempio, il numero 12345678H viene allocato in memoria nel modo seguente:

Indirizzo di memoria	Valore		
0000 0011	12 (MSB)	word	doubleword
0000 0010	34		
0000 0001	56	word	
0000 0000	78 (LSB)		

QUADWORD

Se una doubleword non permette di avere la precisione desiderata, è possibile ricorrere ad una quadword (quattro word) per la memorizzazione di numeri e di stringhe di caratteri di grandi dimensioni. Lo schema di allocazione in memoria di una quadword, illustrato in Figura 1.6, ricalca gli schemi già

Indirizzo di memoria	Valore	
0000 0111	12H (MSB)	doubleword 32 bit
0000 0110	34H	
0000 0101	56H	
0000 0100	78H	
0000 0011	90H	doubleword 32 bit
0000 0010	ABH	
0000 0001	CDH	
0000 0000	EFH (LSB)	

Figura 1.6 Rappresentazione di una quadword in memoria

esaminati per una word e una doubleword. La doubleword meno significativa di una quadword viene memorizzata nella locazione di indirizzo inferiore, mentre la doubleword più significativa viene allocata in posizione consecutiva, verso gli indirizzi di memoria crescenti. La Figura 1.6 indica come avviene la memorizzazione del numero 1234567890ABCDEF.

TENBYTE

Una tenbyte (decina di byte, cioè 80 bit) è in grado di memorizzare numeri e stringhe di caratteri di grandi dimensioni e rappresenta il tipo di dato più esteso che sia possibile definire per il microprocessore 80286. Lo schema di allocazione in memoria di una tenbyte è identico agli schemi esaminati precedentemente per la memorizzazione di una doubleword e di una quadword: il byte meno significativo dei dieci è il primo ad essere allocato, seguito dagli altri byte, in posizione consecutiva e verso gli indirizzi di memoria crescenti, fino al byte più significativo.

TIPI DI DATI DEFINITI NELL'80386

In aggiunta ai tipi di dati definiti dall'8088, dall'8086 e dall'80286, l'architettura dell'80386 supporta gli interi, con o senza segno, fino a 32 bit. Inoltre, la dimensione del tipo puntatore viene estesa a 32 bit solo di offset e a 48 bit complessivamente. L'architettura dell'80386 supporta due tipi di strutture dati non definibili sull'80286 – DBIT (un bit) e DP (48 bit) – per una più semplice gestione delle variabili (si veda a questo proposito il Capitolo 6).

CAMPI DI BIT DI DIMENSIONE NON STANDARD

Anche se la maggior parte dei dati che il programmatore definisce e gestisce può venire descritta con uno dei tipi trattati in precedenza, esistono delle eccezioni. Ad esempio, nel sistema operativo PC-DOS è stato assegnato un campo unico di 16 bit alla variabile "data", composta dai tre campi "anno", "mese" e "giorno", in modo da ottimizzare la quantità di memoria utilizzata. Infatti, sono stati riservati 7 bit (128 configurazioni distinte) per la memorizzazione dell'anno, a partire dal 1980; 4 bit (16 configurazioni distinte) per la memorizzazione di uno dei 12 mesi e 5 bit (32 configurazioni distinte) per la memorizzazione del giorno.
Altri campi di bit di dimensione non standard si trovano in alcune applicazioni grafiche; ad esempio, un solo bit è sufficiente a rappresentare un punto luminoso sul video: se il bit è a 0 il punto non è visibile, mentre se è a 1, il punto è visibile; con due bit, invece, è possibile regolare l'intensità luminosa dei caratteri sul video, ecc.

Tutte queste configurazioni di bit non standard hanno un'importanza decisiva nelle operazioni di memorizzazione, accesso e manipolazione di dati espressi in forma binaria; è indispensabile, in questi casi, gestire il contenuto della memoria in maniera più flessibile e accurata di quanto non si faccia con i tipi di dati standard.

1.8 Operazioni binarie

Tutte le operazioni, anche le più semplici somme e sottrazioni, vengono eseguite dal microprocessore sotto forma di operazioni logiche, cioè – in ultima analisi – di confronti ripetuti su coppie di bit degli operandi.
Queste operazioni logiche – o di confronto – sono molto semplici dal momento che sono possibili solo tre situazioni: entrambi i bit valgono 0, uno solo dei due bit vale 1 oppure entrambi i bit valgono 1. Ogni operazione di somma, sottrazione, moltiplicazione e divisione, prevede quindi la ripetizione, secondo un ordine ben preciso, di operazioni logiche su una delle tre situazioni precedenti.
Nel caso di un'operazione di somma, vengono applicate le seguenti regole, a partire dai bit meno significativi degli operandi:

- La somma di due bit a 0 azzera il bit di Somma
- Se uno dei due bit è a 1, il bit di Somma assume il valore 1
- Se entrambi i bit sono a 1, il bit di Somma viene azzerato e il riporto (Carry) assume il valore 1.

Queste regole di confronto vengono ripetute su ogni coppia di bit degli operandi, tenendo conto del riporto generato dalla coppia di bit adiacente meno significativa. L'unità centrale di elaborazione (Central Processing Unit: CPU) memorizza in un registro interno il risultato di ogni confronto intermedio e fornisce, alla fine, il valore binario risultante.
Oltre alle operazioni di addizione, sottrazione, moltiplicazione e divisione eseguite da un punto di vista logico, il linguaggio assemblatore dispone di vere e proprie istruzioni logiche, come ad esempio AND, OR, XOR, Shift Left, Shift Right, Rotate Left, Rotate Right e Complement.
Esaminiamo in particolare le operazioni AND, OR e XOR, associando ai valori 0 e 1, rispettivamente, lo stato False e True dei bit.
L'operazione logica AND confronta due bit: se entrambi sono a 1, il risultato del confronto è 1. Si noti come questa operazione differisca dalla somma binaria, in cui il confronto di due bit a 1 pone il bit di Somma a 0 e il bit di Carry a 1.

AND logico

bit 0	bit 1	risultato
0	0	0
0	1	0
1	0	0
1	1	1

Molto spesso l'utilità dell'operazione AND consiste nell'eliminare la presenza di alcuni bit, all'interno di un valore numerico. Ad esempio, volendo azzerare, prima di eseguire una moltiplicazione o una divisione decimale, i quattro bit più significativi di un numero decimale non compattato (cioè codificato utilizzando un byte per ogni cifra decimale), è sufficiente eseguire una operazione AND con il numero 0000 1111:

	1010 0011	
AND	0000 1111	Maschera
	0000 0011	

L'operazione logica OR confronta due bit e genera come risultato un 1 se uno dei due bit è a 1. L'operatore OR viene utilizzato per modificare il contenuto di alcuni bit all'interno di un numero.

OR logico

bit 0	bit 1	risultato
0	0	0
0	1	1
1	0	1
1	1	1

Ad esempio, volendo porre a 1 il bit più significativo di un valore numerico di 8 bit, è sufficiente eseguire una operazione OR con il numero 1000 0000.

	0001 1010
OR	1000 0000
	1001 1010

L'operazione di OR esclusivo (XOR) confronta due bit e restituisce come risultato un 1 se i due bit hanno valore diverso. Questa operazione logica è utile quando è necessario complementare alcuni bit all'interno di un numero, come accade spesso nelle applicazioni grafiche.

OR esclusivo

bit 0	bit 1	risultato
0	0	0
0	1	1
1	0	1
1	1	0

Il prossimo esempio esegue il complemento dei quattro bit posti in posizione intermedia all'interno di un numero di 8 bit; è sufficiente in questo caso eseguire una operazione di OR esclusivo con il numero 0011 1100:

	1010 0110
XOR	0011 1100
	1001 1010

Le operazioni logiche Shift Left/Right (SHL e SHR), Rotate Left/Right e Complement si applicano a operandi singoli. Le istruzioni SHIFT costituiscono un'eccellente alternativa all'operazione di divisione e moltiplicazione di un numero per due, in quanto richiedono un numero minore di byte di codice e di cicli di memoria rispetto alle istruzioni esplicite di moltiplicazione e divisione.
Disponendo di un operando senza segno, l'istruzione SHL trasla i bit del numero di una posizione a sinistra, inserendo uno 0 nel bit meno significativo e moltiplicando quindi per due il valore del numero.

SHL	0100 0001	(65 decimale)
	1000 0010	(130 decimale)

La divisione per due di un numero senza segno viene realizzata, invece, dall'istruzione SHR che trasla i bit a destra di una posizione e inserisce uno 0 nel bit più significativo.

SHR	0000 1010	(10 decimale)
	0000 0101	(5 decimale)

Le istruzioni Rotate permettono di cambiare posizione ai bit dell'operando a cui vengono applicate. Diversamente dalle operazioni Shift, che perdono il primo bit da loro spostato a destra o a sinistra, l'istruzione Rotate trasla il bit, lo recupera e lo inserisce nuovamente all'estremo opposto, nella posizione che è rimasta vacante:

ROR	0000 1111	(Rotate Right)
	1000 0111	

ROL	0000 1111	(Rotate Left)
	0001 1110	

1.9 Tecniche di indirizzamento

Le istruzioni dell'80286/80386 non contengono solo informazioni sul tipo di operazione da eseguire, ma anche sul tipo di operandi che manipolano e sulle relative locazioni di memoria. Esistono otto principali modi di indirizzamento:

1. Indirizzamento immediato
2. Indirizzamento a registro
3. Indirizzamento diretto
4. Indirizzamento indiretto con registro
5. Indirizzamento con registro base
6. Indirizzamento diretto con registro indice
7. Indirizzamento con registri base ed indice, con o senza spiazzamento
8. Estensioni dell'80386.

INDIRIZZAMENTO IMMEDIATO

Il microprocessore decodifica il modo di indirizzamento direttamente dalla sintassi dell'istruzione. Ad esempio, se l'istruzione è del tipo:

```
MOV    AH,00
MOV    AL,04
```

si tratta di indirizzamento immediato, in quanto l'operando è un valore numerico. In questo caso, infatti, il registro AH viene caricato con il valore nullo (00), mentre il registro AL viene caricato con il valore binario 0000 0100. La seguente istruzione realizza il trasferimento, nel registro AX, di un dato di 16 bit, espresso nel formato esadecimale (ogni numero esadecimale che inizia con una lettera deve essere sempre preceduto dallo 0):

```
MOV    AX,0FFFFH
```

Nel modo di indirizzamento immediato, la dimensione in bit dell'operando sorgente viene adattata a quella dell'operando destinazione, cioè il bit più significativo dell'operando sorgente viene replicato fino a quando il numero di bit dei due operandi risulta uguale.
Ad esempio:

```
MOV    AX,302
```

Questa istruzione realizza il trasferimento del valore binario di 10 bit (0100101110), equivalente al numero 302, nel registro AX di 16 bit; viene in-

serito automaticamente (da parte del programma assembler) nei 6 bit più significativi del registro AX, non interessati dal trasferimento, il valore del bit più significativo (0) del numero 0100101110; dunque il contenuto finale del registro AX è 0000000100101110.

```
MOV    AL,-40
```

L'operazione di estensione del bit di segno si applica anche a operandi di 8 bit. Nell'esempio precedente, la rappresentazione a 7 bit del numero −40, cioè 1011000, viene estesa nella rappresentazione a 8 bit, cioè 11011000, essendo 8 la dimensione in bit dell'operando destinazione (registro AL).

INDIRIZZAMENTO A REGISTRO

Nell'indirizzamento a registro, anche il valore dell'operando sorgente è contenuto in uno dei registri interni dell'80286/80386 e la sua dimensione può essere di 8 bit, di 16 bit, oppure, nel caso dell'80386, anche di 32 bit. L'assembler valuta la dimensione in bit dell'operando in base al tipo di registro utilizzato.
Ad esempio:

```
MOV    DS,AX
```

In questo caso, l'indirizzamento con registro prevede che l'operando sorgente (16 bit) sia contenuto nel registro AX e l'istruzione effettua il suo trasferimento nel registro DS. Un'istruzione può coinvolgere anche registri sorgente e destinazione di soli 8 bit, come nel seguente esempio:

```
MOV    DL,AL
```

L'indirizzamento immediato e l'indirizzamento a registro costituiscono i due modi di indirizzamento meno dispendiosi in termini di cicli macchina necessari alla loro realizzazione; infatti, nel primo caso gli operandi sono parte integrante dell'istruzione stessa, mentre, nel secondo caso, sono già contenuti nei registri di CPU e quindi non occorre effettuare alcun accesso a memoria o a dispositivi esterni, con conseguente diminuzione del tempo di esecuzione.
I restanti sei modi di indirizzamento richiedono un tempo di esecuzione maggiore, in quanto il microprocessore deve calcolare l'indirizzo dell'operando tramite l'indirizzo base del segmento che lo contiene e il suo spiazzamento all'interno del segmento (queste due informazioni sono indicate esplicitamente oppure sono contenute a loro volta in altrettanti registri). L'indirizzo così calcolato rappresenta l'indirizzo effettivo dell'operando e viene indicato simbolicamente con la notazione EA (Effective Address).

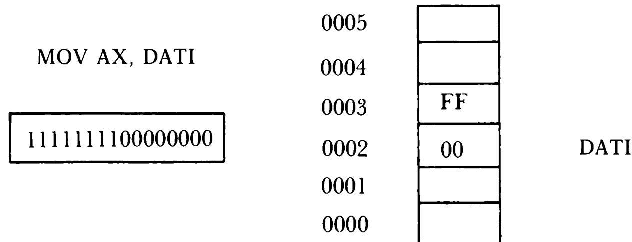


Figura 1.7 Indirizzamento diretto

INDIRIZZAMENTO DIRETTO

Nell'indirizzamento diretto, lo spiazzamento (offset) dell'operando all'interno del segmento è parte integrante dell'istruzione e viene espresso con un valore di 16 bit. Questo spiazzamento viene sommato al contenuto, precedentemente traslato a sinistra di 4 bit, del registro DS, così da ottenere l'indirizzo effettivo di 20 bit dell'operando (indirizzo fisico). Generalmente, nel caso di indirizzamento diretto, l'operando è una etichetta (label).
Ad esempio, l'istruzione indicata nella Figura 1.7 forza il microprocessore a caricare il registro AX con il contenuto della locazione di memoria referenziata dall'indirizzo fisico associato all'etichetta DATI. Si tenga presente che il microprocessore memorizza il byte meno significativo di una word a indirizzi inferiori e il byte più significativo in posizione consecutiva, verso indirizzi crescenti.

INDIRIZZAMENTO INDIRETTO CON REGISTRO

Nell'indirizzamento indiretto con registro, invece di referenziare con una etichetta l'indirizzo dell'operando sorgente, la locazione di memoria in cui è contenuto il valore dell'operando viene calcolata a partire dal suo spiazzamento, che è memorizzato in uno dei seguenti registri: SI (indice sorgente), DI (indice destinazione), BX (registro base), oppure, in alcuni casi, BP (puntatore alla base).
Il programma assembler riconosce, dalla sintassi dell'istruzione, l'indirizzamento indiretto con registro. Infatti, per indicare che il valore dell'operando sorgente è interno alla cella di memoria, il cui spiazzamento è memorizzato in uno dei precedenti registri, si utilizza simbolicamente una coppia di parentesi quadre, [ ], che contengono l'identificatore del registro

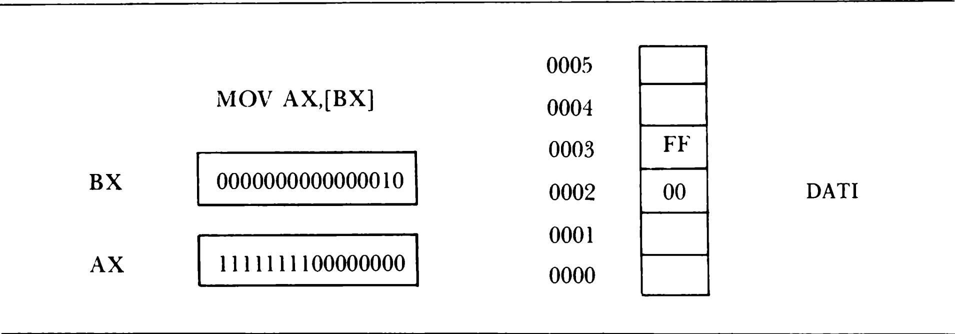


Figura 1.8 Indirizzamento indiretto con registro

stesso. Nella Figura 1.8, l'istruzione viene correttamente eseguita se il registro BX contiene già lo spiazzamento della locazione di memoria identificata simbolicamente da DATI. Dunque, per prima cosa, è indispensabile caricare nel registro BX lo spiazzamento, con un'istruzione di questo tipo:

```
MOV    BX,OFFSET DATI
```

dove OFFSET è un operatore che restituisce lo spiazzamento dell'operando a cui viene applicato.
Esiste anche l'istruzione LEA (Load Effective Address) che permette di caricare, nel registro BX, lo spiazzamento di DATI:

```
LEA    BX,DATI
```

L'indirizzamento indiretto con registro può essere utilizzato per referenziare i dati che sono memorizzati secondo un formato tabellare. In questo modo, l'accesso ai singoli valori diventa più efficiente rispetto ad un'operazione esplicita che calcola l'indirizzo di memoria di ogni dato in tabella.

INDIRIZZAMENTO RELATIVO CON REGISTRO BASE

L'indirizzamento relativo con registro base prevede che l'indirizzo effettivo di un operando, all'interno del segmento selezionato, venga determinato sommando uno spiazzamento al contenuto di un registro base (BX oppure BP). Questo tipo di indirizzamento è utilizzato spesso per accedere a strutture dati complesse, come ad esempio i record. Il registro base punta alla base della struttura dati e lo spiazzamento permette di accedere al campo che interessa. Cambiando valore allo spiazzamento, si può accedere a tutti i campi del record, mentre per referenziare lo stesso campo, in record diversi, è sufficiente modificare il contenuto del registro base.

```
.
.
.
MESGE1    DB    'IL LINGUAGGIO ASSEMBLATORE È VELOCE', '$'
MESGE2    DB    'ED EFFICIENTE','$'
.
.                                        Data
.                                       segment
          LEA   BX,MESGE1          0009  | 47 |   G
.                                  0008  | 41 |   A
.                                  0007  | 55 |   U
.
          MOV   AL,[BX]+4          0006  | 47 |   G
                                   0005  | 4E |   N
          BX   [0000000000000000]  0004  | 49 |   I
                                   0003  | 4C |   L
                                   0002  | 20 |   b
                                   0001  | 4C |   L
                                   0000  | 49 |   I
                AL  [4D]           M                      MESGE1
```

Figura 1.9 Indirizzamento relativo con registro base

Esaminando le istruzioni indicate nella Figura 1.9 si può facilmente constatare che la variabile MESGE1 contiene una stringa di caratteri. L'istruzione LEA carica, nel registro BX, il valore dello spiazzamento della variabile MESGE1, per cui, per accedere al quarto elemento di MESGE1, basta sommare l'indirizzo di base (BX) della variabile MESGE1 con lo spiazzamento +4.
L'assemblatore riconosce tre rappresentazioni equivalenti di indirizzamento con base:

```
LEA    [BX]+4
LEA    4[BX]
LEA    [BX+4]
```

La prima rappresentazione è la più frequente, ma è corretto anche anteporre (o sommare in parentesi quadre) all'identificatore del registro di base il valore dello spiazzamento.
Per accedere a tutti gli elementi di MESGE1 è sufficiente incrementare il valore dello spiazzamento, mentre per referenziare il contenuto di un'altra variabile, è indispensabile caricare, nel registro BX, l'indirizzo di base, ad esempio con la seguente istruzione:

```
LEA    BX,MESGE2
```

INDIRIZZAMENTO DIRETTO CON REGISTRO INDICE

Nell'indirizzamento diretto con registro indice, la posizione di memorizzazione dell'operando, all'interno del segmento selezionato, viene calcolata sommando uno spiazzamento al contenuto di un registro indice (SI o DI). Questo tipo di indirizzamento è utile per accedere agli elementi di un array statico. Il valore dello spiazzamento identifica la posizione di inizio dell'array, mentre il contenuto del registro indice seleziona un elemento all'interno della struttura dati. Diversamente dai record, i cui campi possono variare nel tipo di dati memorizzati e nella dimensione, gli elementi dell'array sono omogenei; dunque, per accedere agli elementi di un array è sufficiente incrementare o decrementare il valore dello spiazzamento, rispetto all'indirizzo di base dell'array.
Ad esempio:

```
MOV    SI,4
MOV    AL,ARRAY1[SI]
```

Il programmatore deve avere cura di scegliere un appropriato valore di spiazzamento, a seconda del tipo di elementi di cui si compone l'array. Le precedenti istruzioni caricano il registro AL con il quinto valore di ARRAY1. Come risulta chiaro dalla Figura 1.10, a seconda del tipo di dati di cui si compone l'array è possibile ad esempio caricare il registro AX con un dato di 16 bit, con un'istruzione del tipo:

```
MOV    AX,ARRAY1[SI]
```

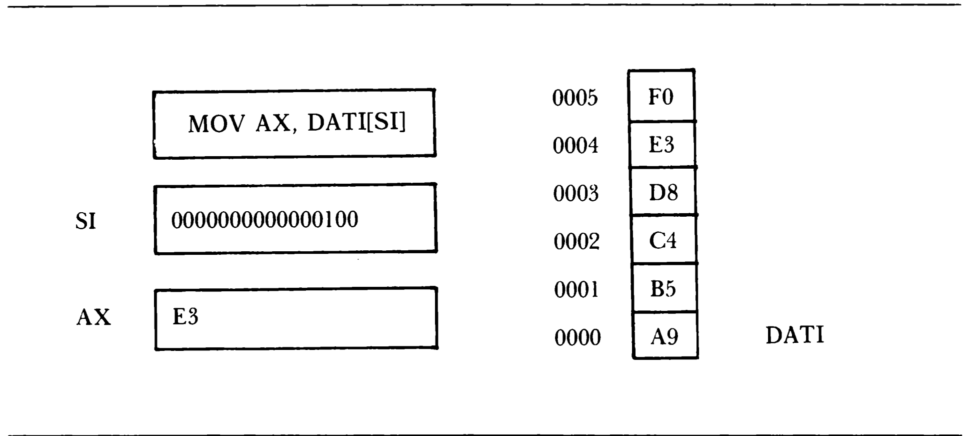


Figura 1.10 Indirizzamento diretto con registro indice

INDIRIZZAMENTO CON REGISTRI BASE E INDICE

In questo caso, lo spiazzamento dell'operando, all'interno del segmento selezionato, viene determinato sommando il contenuto del registro base con il contenuto del registro indice e, a volte, con uno spiazzamento. Senza spiazzamento, l'indirizzamento con base e indice viene di solito usato per accede-

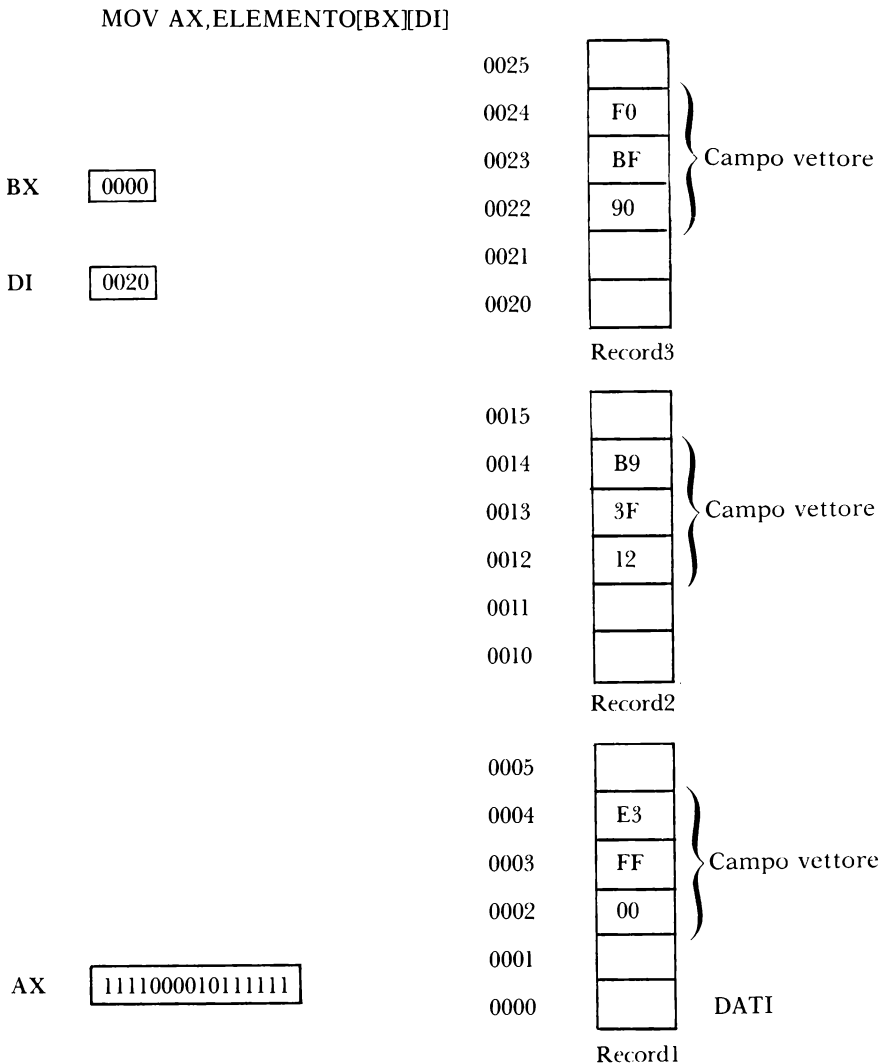


Figura 1.11 Indirizzamento con registri base e indice

re agli elementi di un array dinamico (cioè un array il cui indirizzo di base può cambiare nel corso dell'esecuzione del programma).
Utilizzando lo spiazzamento, invece, è possibile accedere ai singoli elementi di un array, dove l'array costituisce un campo di una struttura dati più complessa, come ad esempio un record.
In quest'ultimo caso, come viene indicato nella Figura 1.11, il registro base punta alla base della struttura record, il registro indice (DI) contiene la distanza del campo array dalla posizione di inizio del record, mentre lo spiazzamento dell'elemento, all'interno dell'array selezionato, è contenuto nella variabile ELEMENTO (supponiamo che questo valore sia 03).
Nell'esempio di Figura 1.11, l'indirizzo di base della struttura record è 0000 ed è memorizzato nel registro BX; per accedere al terzo record della struttura dati, è necessario disporre dello spiazzamento 0020, contenuto in DI, mentre lo spiazzamento del terzo elemento del campo array del terzo record corrisponde al valore iniziale della variabile ELEMENTO.

ESTENSIONI DELL'80386

Nel microprocessore 80386, vengono realizzati modi di indirizzamento a 32 bit, nei quali qualunque registro può essere utilizzato come registro base o come registro indice. Questi modi di indirizzamento richiedono che gli eventuali registri base o indice utilizzati contengano valori di 32 bit. Qualunque istruzione che opera su modi di indirizzamento a 16 bit tronca il contenuto dei registri di 32 bit, in quanto ignora i 16 bit più significativi.

1.10 Stile di programmazione

Un programma scritto in linguaggio assemblatore consiste di una sequenza di istruzioni che indicano all'assemblatore le azioni che il microprocessore deve compiere. Questa successione di istruzioni costituisce il codice sorgente. Come ogni altro linguaggio di programmazione, anche il linguaggio assemblatore possiede una sintassi.
In particolare, una generica istruzione in linguaggio assemblatore è composta da quattro campi:

campo nome *campo operatore* *campo operando* *campo commento*

Alcune istruzioni non utilizzano tutti i quattro campi. Il campo commento è opzionale, ma la sua presenza è indispensabile per documentare alcuni passaggi del programma, che, altrimenti, non sarebbero di immediata interpretazione con la sola lettura del codice.

CAMPO NOME

Il campo nome, spesso indicato come campo etichetta (label), assegna un identificatore simbolico all'indirizzo fisico di inizio dell'istruzione. In questo modo, il programmatore può referenziare l'istruzione con un nome, senza la necessità di conoscere il suo indirizzo effettivo in memoria. Questo accorgimento risulta particolarmente utile per generare codice rilocabile: il linker (letteralmente "collegatore") risolve i riferimenti simbolici, sostituendo ad ognuno di essi l'indirizzo fisico di memoria in cui è stata allocata l'istruzione referenziata, e ogni eventuale modifica di questo indirizzo non influenza minimamente il codice sorgente. Anche se il programmatore può dare un nome ad ogni istruzione, questo campo viene di solito riservato a quelle istruzioni che non vengono referenziate sequenzialmente, come avviene ad esempio nel caso di cicli, di salti, di chiamate di procedure; il campo nome viene anche utilizzato per identificare una variabile, una costante oppure un segmento di codice o di dati.
Un nome deve iniziare con un carattere alfabetico e deve avere una estensione massima di 31 caratteri, dei quali fanno parte:

- Tutte le lettere dalla A alla Z
- Le cifre da 0 a 9
- I simboli speciali _ $? @ %

Il programmatore deve fare molta attenzione nella scelta di un nome; questo, infatti, non deve mai coincidere con una parola riservata o con una direttiva del linguaggio assemblatore e può contenere il punto (.) solo come primo carattere.

Variabili

Un nome di variabile rappresenta una locazione di memoria che è accessibile nel programma e il cui contenuto può cambiare nel corso dell'esecuzione del programma stesso. La definizione di una variabile consiste nell'indicare il suo indirizzo fisico, il tipo di dato ad essa associato e la sua dimensione. Il programmatore può utilizzare le variabili come operandi di tipo semplice o strutturato.

Etichette (label)

I nomi, detti anche etichette (label) in questo caso, referenziano istruzioni eseguibili e possiedono tre attributi: un indirizzo di segmento, un offset interno al segmento e un indicatore che definisce il tipo di accesso (NEAR o FAR).
Il microprocessore può indirizzare una etichetta in due modi distinti: se l'etichetta è interna al segmento di codice in esecuzione (etichetta NEAR), viene utilizzato il solo campo offset per risolvere il riferimento. La definizione

di una etichetta NEAR implica la presenza di un carattere due punti (:), immediatamente dopo l'etichetta, oppure, con notazione esplicita, si può utilizzare la direttiva NEAR:

```
LOOP1:
```

Il carattere due punti (:) collocato nella posizione indicata informa l'assemblatore che la corrispondente istruzione viene referenziata nello stesso segmento di codice in cui compare l'etichetta.

```
CONT     LABEL     NEAR
```

In questo caso, l'etichetta CONT viene indicata esplicitamente come NEAR tramite la direttiva LABEL.
Il secondo modo di indirizzamento di una etichetta richiede sia l'indirizzo di segmento sia l'offset all'interno del segmento. In questo caso, l'istruzione che viene referenziata non fa parte dello stesso segmento di codice e l'etichetta che la identifica simbolicamente è di tipo FAR.

```
CODICE     LABEL     FAR
```

In questo esempio, viene indicato esplicitamente per l'etichetta (tramite la direttiva LABEL) l'attributo FAR, che può anche essere usato per definire costanti, procedure e riferimenti esterni, come viene indicato nei seguenti esempi:

```
DIECI     EQU     FAR     10
PROCEDURA     PROC     FAR
EXTRN     RIFERIMENTO:FAR
```

Costanti
In linguaggio assemblatore può essere necessario associare un nome a una locazione di memoria che deve essere inizializzata con un determinato valore, oppure associare al nome stesso il significato di notazione mnemonica di un valore costante.
I modi in cui si specifica il valore costante e lo si associa al nome corrispondente sono descritti di seguito.

Costanti binarie: sono costituite da una sequenza di 0 e 1 seguita dalla lettera B. Ad esempio:

```
OTTO     EQU     00001000B
```

Costanti decimali: sono costituite da una sequenza di cifre comprese tra 0 e 9, facoltativamente seguite dalla lettera D. In linguaggio assemblatore, una

sequenza di sole cifre rappresenta sempre un numero decimale a meno che la direttiva RADIX non abbia modificato la base. Un esempio di costante decimale è il seguente:

```
QUARANTA    EQU    40D
```

Costanti esadecimali: sono costituite da una sequenza di cifre comprese tra 0 e 9 e di lettere comprese tra A e F, seguite dalla lettera H. Il primo carattere deve comunque essere una cifra per segnalare all'assembler che il valore rappresenta un numero e non una etichetta o il nome di una variabile. Se il valore esadecimale inizia con una lettera, perché l'assembler lo interpreti correttamente, deve essere preceduto da uno 0. La dichiarazione di una costante esadecimale è quindi del tipo:

```
CINQUANTA    EQU    32H
NUMESAD      EQU    0FFH
```

In questo ultimo esempio, la cifra più significativa è stata preceduta dallo 0, in modo che l'assemblatore sappia che FFH rappresenta un numero esadecimale e non una etichetta o il nome di una variabile.

Costanti ottali: contengono una sequenza di cifre comprese tra 0 e 7 e sono seguite dalla lettera O o dalla lettera Q. Ad esempio:

```
SEI    EQU    6O oppure 6Q
```

Stringhe di caratteri: possono contenere qualunque carattere ASCII racchiuso tra apici singoli o doppi. Se si deve inizializzare una zona di memoria con una costante costituita da più di due caratteri, si deve utilizzare la direttiva DB (Define Byte). Se la stringa di caratteri contiene solo uno o due caratteri, possono essere usate anche le direttive DD, DQ, DT o DW. Ad esempio:

```
INIZ    DD    'B'
NOME    DB    "PAOLO ROSSI"
```

Costanti in virgola mobile (floating point): questo tipo di costanti viene espresso in notazione scientifica decimale e non è supportato dalla versione ridotta del programma assemblatore della IBM. Ad esempio:

```
SENO    DD    0.332E-1
```

Costanti reali esadecimali: sono costituite da una sequenza di cifre comprese tra 0 e 9 e di lettere tra A e F, seguite dalla lettera R. Come le costanti esadecimali, il primo carattere deve essere una cifra e, inoltre, ogni costante reale esadecimale deve contenere un numero di cifre pari a 8, 16 o 20, a

meno che la prima cifra non sia 0. In questo ultimo caso, il numero totale di cifre deve essere maggiore di una unità rispetto ai valori precedenti (cioè 9, 17 o 21). Questo tipo di dati non è supportato dalla versione ridotta del programma assemblatore della IBM. Ad esempio:

```
NUMESADR    EQU    0FAB12345R
```

Rappresentazione mnemonica di valori costanti: è possibile assegnare una etichetta (campo nome) al valore di un'epressione (campo operando), utilizzando la direttiva EQU o il simbolo di uguaglianza (=). Nel primo caso, all'etichetta viene assegnato un valore costante che non può essere alterato nel resto del programma, mentre nel secondo caso alla stessa etichetta si può assegnare, in un differente punto del programma sorgente, un diverso valore costante. Ad esempio:

```
ELEM    EQU     [BP+16]
BASE    = 1980
```

Nel primo esempio (ELEM EQU [BP+16]), l'identificatore ELEM può sostituire l'espressione [BP+16]. Allo stesso modo, BASE può sostituire il valore 1980, ma a BASE può anche essere assegnato un nuovo valore nel resto del programma.

Nomi di segmenti: viene assegnata nel campo nome della direttiva SEGMENT una etichetta che referenzia il segmento. Ad esempio:

```
CODICE    SEGMENT    PARA    'CODE'
```

CAMPO OPERATORE

Il campo operatore contiene un codice mnemonico, costituito da un numero di caratteri variabile tra due e sei, che indica simbolicamente il tipo di istruzione. Il codice mnemonico è l'abbreviazione di una parola in lingua inglese, allo scopo di rendere il codice sorgente più facilmente leggibile; al codice mnemonico corrisponde generalmente un codice macchina binario, la cui codifica è definita in una tabella di conversione interna. Un operatore, o codice mnemonico, può rappresentare anche una macroistruzione oppure una direttiva. Ad esempio:

```
INIZIO    MOV    AX,0H
```

INIZIO è una etichetta e MOV rappresenta l'operazione da eseguire. Dopo il campo operatore segue sempre il campo operando; ogni operatore non so-

lo dice all'assemblatore quale istruzione deve eseguire, ma anche quanti operandi occorrono per realizzarla.
Un operatore può anche contenere un riferimento ad una macro; in questo caso, si chiede all'assembler di tradurre in codice macchina una sequenza predefinita di codice, come se questa fosse esplicitamente presente nel programma in luogo dell'invocazione della macro. Ad esempio:

```
INT_DOS     MACRO     SERVIZIO
```

Questa direttiva informa l'assemblatore che il codice seguente costituisce la definizione di una MACRO. Una direttiva di solito non produce codice macchina, ma forza l'assemblatore ad eseguire alcune operazioni sui dati, sul codice, sulle macro.

CAMPO OPERANDO

Il campo operando contiene i riferimenti alle locazioni in cui sono memorizzati i dati che vengono usati dall'istruzione. Se l'istruzione richiede uno o due operandi, questi devono essere separati dal campo operatore con almeno uno spazio; se gli operandi sono due, devono essere tra loro separati con una virgola (,) (si noti che esistono anche istruzioni che non richiedono alcun operando).
Quando un'operazione coinvolge due operandi, il primo di essi viene detto operando destinazione, mentre il secondo operando sorgente. Esempi di istruzioni che richiedono due operandi sono i trasferimenti di dati tra i registri e le locazioni di memoria. Ad esempio:

```
MOV     AX,8
```

In questo caso, il valore dell'operando sorgente (8) è parte integrante dell'istruzione e viene trasferito nel registro AX, che costituisce l'operando destinazione.

CAMPO COMMENTO

Il campo commento è l'ultimo, ma non il meno importante, dei quattro campi di cui si compone un'istruzione e permette al programmatore di documentare internamente il codice sorgente. I commenti vengono ignorati dall'assemblatore, ma compaiono nel listato del programma sorgente. Perché un commento si riferisca ad un'istruzione, deve essere separato da essa con almeno uno spazio e iniziare con un punto e virgola (;). L'utilità del com-

mento è quella di descrivere il significato di quelle linee di codice che non sono immediatamente comprensibili. Ad esempio:

```
MOV    AH,45H    ;parametro per la lettura di un carattere
```

Come indicato, il commento spiega perché il registro AH viene caricato con il valore 45H.

1.11 Un esempio di programma in linguaggio assemblatore

L'esempio di Figura 1.12 rappresenta un programma nel linguaggio assemblatore, dopo che è stato assemblato. Alla destra è visualizzato il codice sorgente che è facilmente leggibile e ben documentato. Ovviamente, per i

```
IBM Personal Computer MACRO Assembler   Version 2.00    Page 1-1
                                        09-19-85

                    page ,132
                    ;per macchine 8088/80386
                    ;programma che esegue una semplice addizione
                    ;esadecimale con indirizzamento immediato

0000                STACK   SEGMENT  PARA STACK
0000   40 [         DB      64 DUP ('MYSTACK ')
          4D 59 53 54
          41 43 4B 20
                            ]

0200                STACK   ENDS

0000                CODICE     SEGMENT  PARA 'CODICE' ;definisce il segmento di codice
0000                PROCEDURA  PROC   FAR            ;inizio della procedura
                    ASSUME     CS:CODICE,SS:STACK
0000   1E           PUSH       DS                    ;salva DS sullo stack
0001   2B C0        SUB        AX,AX                 ;azzera AX
0003   50           PUSH       AX                    ;salva 0 sullo stack

                    ;somma di tre numeri usando l'indirizzamento immediato
0004   B0 23        MOV        AL,23H                ;in AL c'è 23H
0006   04 0A        ADD        AL,0AH                ;somma 0AH ad AL
0008   04 10        ADD        AL,10H                ;somma 10H ad AL
                    ;fine della somma, in AL c'è il risultato

000A   CB           RET                              ;controllo al DOS
000B                PROCEDURA  ENDP                  ;fine della procedura
000B                CODICE     ENDS                  ;fine del segmento di codice

                    END                              ;fine del programma
```

Figura 1.12 Un programma assemblato

programmatori non ancora esperti, il codice macchina generato risulta, invece, di difficile interpretazione.
La prima colonna (numeri da 1 a 31) contiene l'indicazione del numero di linea e viene inserita dall'assembler. In seconda colonna è visualizzato l'indirizzo della locazione di memoria di inizio, in cui è memorizzato l'operatore mnemonico tradotto. Questi indirizzi a 16 bit sono espressi nel formato esadecimale. La terza e quarta colonna rappresentano la traduzione in codice macchina dell'operatore mnemonico. Il contenuto di queste colonne dipende dal tipo di istruzione e dal numero di operandi richiesti.
Le restanti colonne riportano, senza modifiche di formato, il programma sorgente nel linguaggio assemblatore e contengono un campo nome, un campo operatore, un campo operando e un campo commento (quest'ultimo è opzionale). Si noti che le linee 2, 3, 4, 21 e 25 sono solamente commenti. Nel Capitolo 5 viene spiegato dettagliatamente come scrivere un programma in linguaggio assemblatore.

2

Introduzione agli assembler

Questo capitolo è dedicato all'esame delle caratteristiche e delle funzioni di un programma assembler. In particolare, vengono evidenziate le differenze e le similitudini esistenti tra gli assembler – programmi, cioè, che generano codice eseguibile a partire da un codice sorgente scritto nel linguaggio assemblatore – e i compilatori, che generano la versione eseguibile dei programmi scritti in un linguaggio di alto livello.
L'esempio presentato ha lo scopo di indicare i passi che sono indispensabili per creare, tradurre ed eseguire un semplice programma scritto in linguaggio assemblatore, indicando anche funzioni e scopi di alcune direttive fondamentali che devono essere incluse nel codice sorgente.
Il programma ESEMPIO.ASM, che viene descritto in questo capitolo, introduce già il lettore alla conoscenza della sintassi del linguaggio assemblatore (per avere maggiori informazioni sul set di istruzioni dell'80286/80386 e dell'80287/80387 vedere i Capitoli 3 e 4).
Un programma assembler legge un file di testo (*codice sorgente*) che si compone di *codici mnemonici* – cioè di comandi espressi come parole abbreviate in lingua inglese – e li traduce in sequenze binarie di 1 e di 0, generando il *codice macchina* del microprocessore.
La rappresentazione delle istruzioni nella codifica mnemonica permette al programmatore di scrivere un codice sorgente sufficientemente leggibile, mentre la corrispondenza biunivoca esistente tra i codici mnemonici e il codice macchina permette all'assembler di realizzare una traduzione efficiente, accurata e particolarmente rapida.
Un compilatore esegue essenzialmente gli stessi passi di un assembler (lettura di un file di testo e traduzione in codice macchina), ma è sicuramente

più semplice programmare in linguaggio di alto livello piuttosto che in linguaggio assemblatore.
Il linguaggio di alto livello, infatti, garantisce al programmatore una completa indipendenza dall'architettura interna della macchina, a differenza del linguaggio assemblatore, le cui istruzioni contengono quasi sempre riferimenti espliciti ai registri, alle locazioni di memoria, alle porte di ingresso e uscita del calcolatore.
Tra il codice sorgente di alto livello e il codice macchina, però, non esiste una corrispondenza biunivoca e spesso, a fronte di una maggiore facilità di programmazione in un linguaggio di alto livello, corrisponde un codice compilato di dimensioni maggiori. Questa constatazione non deve naturalmente portare alla conclusione che i compilatori non siano in grado di generare codice macchina in modo efficiente, ma solo a sottolineare il fatto che la massima efficienza è raggiungibile solo con un uso "esperto" del linguaggio assemblatore.

2.1 Confronto tra linguaggio assemblatore e codice macchina

Le due parti di codice sotto riportate, che rappresentano versioni diverse di alcune semplici operazioni (inizializzazione, caricamento, nel registro accumulatore, del contenuto della cella di memoria NUMUNO e conseguente operazione di somma con il valore numerico 3H) evidenziano l'utilità del programma assembler e dimostrano l'esistenza di una corrispondenza biunivoca tra le istruzioni del linguaggio assemblatore e il codice macchina.

Codice macchina	Linguaggio assemblatore equivalente
00011110	PUSH DS
00101011	SUB AX,AX
11000000	
01010000	PUSH AX
10111000	MOV AX,DATI
00000001	
11101100	
10001110	MOV DS,AX
11011000	
10100001	MOV AX,NUMUNO
0000	
00000101	ADD AX,3H
0011	

La versione in codice macchina di un programma scritto nel linguaggio assemblatore è comunemente conosciuta con il nome di codice oggetto e può

essere direttamente allocata in memoria ed eseguita, dopo aver aggiornato i riferimenti interni agli indirizzi fisici.
Programmare in codice macchina è particolarmente dispendioso in termini di tempo impiegato e i programmi così codificati risultano difficilmente leggibili e sicuramente non esenti da errori. Esaminando il precedente programma in codice macchina, è difficile capire immediatamente se i valori binari rappresentano istruzioni, dati oppure indirizzi. Dunque, non solo risulta estremamente complicato scrivere un programma in codice macchina, ma ugualmente problematica è la sua correzione.
Anche se i microprocessori interpretano solamente sequenze di uni e di zeri, gli assembler permettono di programmare in un linguaggio – linguaggio assemblatore – che è certamente più comprensibile del linguaggio macchina, in quanto le istruzioni non vengono più espresse in un formato numerico di difficile interpretazione, ma tramite comandi alfabetici che favoriscono la leggibilità e la correzione dei programmi.
Nel libro, frequenti sono i riferimenti a tre popolari programmi assembler (IBM Macro Assembler, Microsoft Macro Assembler e Turbo Editasm Assembler), che hanno molte caratteristiche comuni, ma che si differenziano in alcune direttive di traduzione e in qualche comando di editing. Tutti e tre gli assembler usano, comunque, lo stesso set di istruzioni (l'appendice contiene una descrizione dettagliata delle caratteristiche di ognuno dei tre assembler).
A proposito del set di istruzioni, è interessante notare come la filosofia della Intel sia quella di aggiungere nuove funzioni ai microprocessori più recenti, mantenendo però compatibilità con il passato. Ogni nuovo componente introduce quindi nuove istruzioni – che corrispondono a nuove funzioni disponibili – ma è in grado di eseguire (con prestazioni ovviamente superiori) anche gli stessi programmi scritti per i componenti precedenti.

2.2 Sviluppo di un programma nel linguaggio assemblatore

Questo paragrafo discute in dettaglio i passi necessari per scrivere un programma nel linguaggio assembler ed evidenzia le caratteristiche comuni ai tre assemblatori di cui abbiamo parlato nel paragrafo precedente (il Capitolo 5, invece, viene interamente dedicato alla descrizione delle tecniche elementari di programmazione).
Lo sviluppo di un programma nel linguaggio assemblatore comporta (Figura 2.1): l'uso di un editor di testo per scrivere il codice sorgente; l'uso di un assembler che traduca il codice sorgente in codice oggetto (codice macchina); l'uso di un linker per risolvere i riferimenti all'interno del codice oggetto e ottenere così il codice eseguibile (un file, cioè, con l'estensione .EXE).

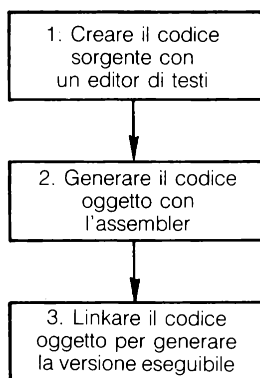


Figura 2.1 Passi necessari per lo sviluppo di un programma in linguaggio assemblatore

Un metodo alternativo al precedente consiste nella creazione di un file .COM, le cui caratteristiche, dipendenti dal tipo di assembler utilizzato, vengono discusse in Appendice B.
Per completare il ciclo di sviluppo di un programma scritto nel linguaggio assemblatore, esiste anche un quarto passo che consiste nella correzione degli eventuali errori sintattici e degli errori che si presentano in fase di esecuzione.

PASSO 1: CREAZIONE DEL CODICE SORGENTE

Per creare il codice sorgente, è possibile utilizzare qualunque editor che generi un file di testo nel formato ASCII. Questo file deve essere privo di qualsiasi codice speciale di controllo, per cui non sono permesse sottolineature, allineamenti, scritte in neretto, soprascritte, sottoscritte, ecc (tutte informazioni superflue per l'assembler).
Molti calcolatori dispongono di semplici editor di linea, come ad esempio l'editor EDLIN disponibile in ambiente DOS. EDLIN è un editor di linea perché gestisce ogni linea di testo singolarmente (è possibile scrivere una sola linea alla volta), a differenza di un editor di video, che permette una modifica più elastica del testo, in quanto l'utente può spostare il cursore in qualunque punto dello schermo, con un solo comando.
Se si intende programmare seriamente nel linguaggio assemblatore, è meglio usare un editor di video. Tutti gli esempi che vengono presentati in questo libro sono stati scritti con l'editor di video Professional Editor della IBM

oppure con il Norton Editor, ma molti word processor (WordPerfect, WordStar e EasyWriter, ad esempio) possono creare file nel formato ASCII, una volta che è stata selezionata la relativa opzione.
La Figura 2.2 illustra un programma in linguaggio assemblatore scritto con un editor; da questo semplice esempio, si possono ricavare le prime indicazioni su come spaziare le linee, a partire dalla prima colonna. Poiché i tre assembler, di cui abbiamo parlato, esaminano solo i file di codice sorgente che presentano l'estensione .ASM, l'identificatore completo, con cui salvare il programma su dischetto, deve essere *nome-programma*.ASM. (Il programmatore non deve riportare nel file anche i numeri di linea, perché vengono automaticamente inseriti dall'editor, per rendere il testo referenziabile).
Esaminiamo ora più dettagliatamente il significato di alcune istruzioni del programma che assumono una particolare importanza.
Le prime due linee di codice, che indicano il nome e la funzione del programma, sono semplici commenti:

```
1. ;programma ESEMPIO per macchine IBM 8088/80286
2. ;il programma visualizza una stringa di caratteri sullo schermo
```

```
 1.;programma ESEMPIO per macchine IBM 8088/80286
 2.;il programma visualizza una stringa di caratteri sullo schermo

 3.STACK   SEGMENT   PARA  'STACK'
 4.        DB        48 DUP ('STACK')
 5.STACK   ENDS

 6.DATI    SEGMENT   PARA  'DATI'
 7.MSG     DB 'IL PROGRAMMA HA PRODOTTO QUESTA STRINGA DI CARATTERI','$'
 8.DATI    ENDS

 9.CODICE  SEGMENT   PARA  'CODICE'    ;definisce il segmento di codice CODICE
10.MAIN    PROC      FAR               ;inizio della procedura MAIN
11.        ASSUME CS:CODICE,DS:DATI,ES:DATI,SS:STACK
12.        PUSH      DS                ;salva il contenuto del registro DS
13.        SUB       AX,AX             ;azzera AX
14.        PUSH      AX                ;salva l'offset 0 sullo stack
15.        MOV       AX,DATI           ;indirizzo di DATI in AX
16.        MOV       DS,AX             ;indirizzo di DATI in DS
17.        MOV       ES,AX             ;indirizzo di DATI in ES

18.;istruzioni che realizzano l'invio del messaggio sul video
19.        LEA       DX,MSG            ;routine DOS per la stampa di una stringa
20.        MOV       AX,09             ;parametro DOS
21.        INT       21H               ;interrupt DOS

22.        RET                         ;il controllo ritorna al DOS
23.MAIN    ENDP                        ;fine della procedura MAIN
24.CODICE  ENDS                        ;fine del segmento di codice CODICE
25.        END                         ;fine del programma
```

Figura 2.2 Il programma ESEMPIO.ASM

e vengono ignorate dall'assembler, come accade per ogni stringa di caratteri che segue un punto e virgola.
La parte di codice successiva, invece, è delimitata da due parole chiave, SEGMENT (linea 3) ed ENDS (linea 5), che indicano all'assembler l'inizio e la fine di un gruppo di istruzioni particolari (in questo caso si tratta della definizione di un segmento di stack):

```
3. STACK   SEGMENT         PARA     'STACK'
4.         DB    48 DUP    ('STACK')
5. STACK   ENDS
```

I macroassembler, come i tre qui considerati, consentono di mantenere attivi in ogni momento fino a quattro segmenti di memoria (il segmento codice, il segmento dati, il segmento stack e il segmento extra) e permettono, quindi, la scrittura di codice sorgente in modo modulare.
La linea 3 contiene la parola chiave STACK (campo etichetta) e specifica il nome che identifica il segmento di stack. L'operando PARA indica che il segmento ha inizio in memoria a partire da un paragrafo di dimensioni standard (16 byte), mentre la stringa 'STACK', alla fine della linea, specifica il nome che il segmento assume in operazioni di *cross-reference*.
La linea 4 definisce la dimensione dello stack, 48 D(efine) B(yte), e indica all'assemblatore di inizializzare la prima di queste locazioni di memoria con l'identificatore simbolico 'STACK'. L'istruzione ENDS (linea 5) contiene una etichetta che deve essere identica a quella presente nella relativa istruzione SEGMENT e che specifica la fine della definizione del segmento.
Le tre linee di codice successive definiscono un segmento dati che viene identificato con il nome DATI:

```
6. DATI   SEGMENT        PARA    'DATI'
7. MSG    DB    'IL PROGRAMMA HA PRODOTTO QUESTA STRINGA DI
                CARATTERI','$'
8. DATI   ENDS
```

Anche questo segmento ha inizio in memoria in corrispondenza di un paragrafo, viene identificato, in operazioni di cross-reference, dal nome simbolico 'DATI' (linea 6) e contiene la sola variabile MSG (linea 7), che rappresenta la stringa di caratteri 'IL PROGRAMMA HA PRODOTTO QUESTA STRINGA DI CARATTERI','$' (ogni carattere occupa un byte di memoria).
Alcuni tipi di assembler richiedono che la definizione del segmento dati venga collocata in precise zone del programma, a seconda del tipo e dell'uso delle variabili in esso contenute. Tutti gli esempi del libro definiscono il segmento dati all'inizio del programma, ma esso può essere posizionato anche dopo il segmento codice (se il segmento dati è in cima al programma e viene stampato un messaggio di errore che lo riguarda, conviene trasferire il segmento alla fine del programma).

La linea 9 definisce l'inizio del segmento di codice CODICE che deve essere allocato in memoria a partire da un paragrafo e che viene identificato, in operazioni di cross-reference, dal nome simbolico 'CODICE'.

```
9. CODICE   SEGMENT  PARA    'CODICE' ;definisce il segmento di
                                       codice CODICE
```

Questa linea, come altre linee del programma, contiene un commento che spiega sinteticamente il significato delle istruzioni.
La successiva linea di codice contiene (campo etichetta) l'identificatore MAIN e, di seguito, la notazione dichiarativa PROC(edura):

```
10. MAIN    PROC     FAR                ;inizio della procedura MAIN
```

Le procedure sono sezioni di codice che possono essere attivate, con istruzioni di chiamata, da punti diversi del programma principale. Ogni volta che una procedura viene chiamata, ha inizio l'esecuzione delle istruzioni che la compongono e, al termine, il controllo ritorna al programma chiamante.
La dichiarazione di procedura è delimitata dalle parole chiave PROC ed ENDP e può contenere anche le direttive NEAR o FAR, per informare l'assemblatore su quale tipo di istruzione di chiamata o di salto sia necessaria per referenziare la procedura.
In molti casi, l'attributo FAR o NEAR regola anche il tipo di ritorno dalla procedura al programma chiamante: se la procedura è NEAR, il microprocessore salva automaticamente sullo stack il contenuto del registro IP, mentre, nel caso di chiamata di una procedura FAR, viene salvato anche il contenuto del registro CS.
Il tipo di istruzioni, che sono contenute nella parte iniziale di un programma (linee comprese tra 1 e 10), non è stabilito a priori, in quanto ogni programma dispone di propri commenti, propri segmenti di stack e di dati, inizializzati e dimensionati secondo le proprie esigenze. Le successive 7 linee di codice (quelle comprese tra 11 e 17), invece, fanno parte di ogni programma, essendo direttive per l'assembler.
Infatti, perché un programma nel linguaggio assemblatore esegua correttamente le sue funzioni e restituisca, senza errori, il controllo al sistema operativo, deve contenere anche parti che – pur non svolgendo funzioni direttamente collegate al problema considerato – devono necessariamente essere presenti. Chiamiamo queste parti "istruzioni di definizione dell'ambiente di lavoro".

Istruzioni di definizione dell'ambiente di lavoro

La direttiva ASSUME invita l'assemblatore a referenziare il segmento codice, il segmento dati, il segmento extra e il segmento stack tramite rispettivamente i registri CS, DS (ES) e SS:

```
11.    ASSUME  CS:CODICE,DS:DATI,ES:DATI,SS:STACK
```

In questo modo, il programmatore non è obbligato a indicare esplicitamente quale registro di segmento viene referenziato dall'istruzione corrente, ma tutti i riferimenti al contenuto dei segmenti di codice, dati e stack vengono automaticamente soddisfatti, specificando solo lo spiazzamento della locazione, interna al segmento, che si intende referenziare.
Le tre istruzioni seguenti servono a salvare il contesto all'atto dell'inizio del programma, per poter cedere nuovamente il controllo al sistema operativo – una volta terminato il programma stesso – in modo corretto.

```
12.   PUSH   DS      ;salva il contenuto del registro DS
13.   SUB    AX,AX   ;azzera AX
14.   PUSH   AX      ;salva l'offset 0 sullo stack
```

La linea 12 memorizza sullo stack (con una tecnica pensata per risparmiare tempo) il contenuto corrente del registro DS; la linea 13 genera un offset di valore 0, che viene anch'esso memorizzato sullo stack, dall'istruzione della linea 14.
L'operazione di salvataggio del vecchio contenuto del registro DS è seguita dal caricamento del registro DS con un nuovo valore. Per caricare, nel registro DS, una variabile, occorrono, infatti, due istruzioni:

```
15.   MOV    AX,DATI   ;indirizzo di DATI in AX
16.   MOV    DS,AX     ;indirizzo di DATI in DS
```

Prima viene caricato, nel registro AX, l'indirizzo di inizio del segmento dati (linea 15) e, successivamente, il contenuto di AX viene trasferito nel registro DS (linea 16).
Una istruzione analoga è necessaria per il caricamento del registro ES:

```
17.   MOV    ES,AX     ;indirizzo di DATI in ES
```

In questo esempio, entrambi i segmenti puntano alla stessa locazione di memoria. Con quest'ultima istruzione termina la parte di programma che contiene le istruzioni di definizione dell'ambiente di lavoro.
Le istruzioni comprese tra le linee 18 e 21 costituiscono il codice che svolge le funzioni effettivamente richieste dal problema:

```
18. ;istruzioni che realizzano l'invio del messaggio sul video
19.   LEA    DX,MSG   ;routine DOS per la stampa di una stringa
20.   MOV    AX,09    ;parametro DOS
21.   INT    21H      ;interrupt DOS
```

In questo caso, le quattro linee di codice preparano ed eseguono la chiamata della procedura che visualizza sullo schermo una stringa di caratteri.

Le ultime istruzioni dell'esempio indicano la fine del programma:

```
22.                RET        ;il controllo ritorna al DOS
23.  MAIN          ENDP       ;fine della procedura MAIN
24.  CODICE        ENDS       ;fine del segmento di codice CODICE
25.                END        ;fine del programma
```

L'istruzione RET (linea 22) estrae l'indirizzo di ritorno dallo stack e, in questo caso, restituisce il controllo al sistema operativo. L'istruzione della linea 23 conclude la sezione di codice riservata alla procedura, mentre con l'istruzione ENDS della linea 24 termina il segmento di codice CODICE. L'istruzione END (linea 25), infine, indica all'assembler la conclusione del programma. Dopo aver scritto il codice sorgente e averlo salvato nel file ESEMPIO.ASM, il programmatore può affrontare il passo 2.

PASSO 2: GENERAZIONE DEL CODICE OGGETTO

A questo punto, l'assembler esamina il codice sorgente e fornisce come risultato la traduzione in codice oggetto. I tre tipi di assembler, di cui si parla in questo libro, possiedono un certo numero di opzioni di traduzione (si consultino a questo proposito le Appendici A, B e C), ma ora a noi interessa solo discutere come sia possibile generare rapidamente un file di codice oggetto (un file, cioè, con l'estensione .OBJ).
Se uno dei tre macroassembler è contenuto nel dischetto inserito nel drive A e il file di codice sorgente ESEMPIO.ASM è memorizzato nel dischetto presente nel drive B (e se B è anche il drive selezionato), potete scrivere da tastiera:

B>A:MASM ESEMPIO;
(per il macroassembler IBM e l'assembler Microsoft)

oppure

B>A:TASMB ESEMPIO;
(se state usando il Turbo Editasm)

Selezionate l'opzione F7 per generare il file .OBJ e premete il tasto A per dare inizio alla traduzione. Il programma TASMB chiederà il nome che intendete dare al file .OBJ (viene assunto per difetto il nome ESEMPIO.OBJ: se siete d'accordo, premete il tasto Y).
Si tenga presente che, ai tre assembler, non è necessario specificare anche l'estensione .ASM del file sorgente, in quanto viene assunta per difetto.
Se l'assembler non trova, nel file ESEMPIO.ASM, nessun errore sintattico, viene generato il file ESEMPIO.OBJ, contenente la traduzione in codice oggetto (potete rendervi conto della creazione di questo file, richiedendo l'im-

magine del direttorio del disco B). Anche se in questo file tutte le istruzioni del programma sono espresse in codice macchina, il loro formato non è ancora adatto perché possano essere caricate in memoria dal sistema operativo ed eseguite dal microprocessore.

PASSO 3: CREAZIONE DEL FILE ESEGUIBILE

Per convertire un file di codice oggetto in un file di codice eseguibile, è necessario utilizzare un programma linker (LINK), che risolva i riferimenti tra i moduli del programma (per eventuali opzioni si consultino le Appendici A, B e C). Descriviamo brevemente i comandi che permettono di creare un file eseguibile .EXE.
Supponiamo che il programma LINK (fornito con l'assembler oppure con il DOS) si trovi nel drive A e che il file ESEMPIO.OBJ si trovi nel drive B (drive selezionato). Per creare il file ESEMPIO.EXE, dovete scrivere da tastiera:

```
B>A:LINK ESEMPIO
```

(sia per il macroassembler IBM che per l'assembler Microsoft e per il Turbo Editasm)

Il risultato del comando LINK è il file eseguibile ESEMPIO.EXE, che viene memorizzato nel dischetto inserito nel drive B. Per eseguire questo programma, scrivete:

```
B>ESEMPIO
```

Dopo che il programma ha visualizzato il suo breve messaggio, riapparirà il messaggio B> del DOS.

3

Architettura: registri, flag, istruzioni

3.1 Il microprocessore 80286

Il microprocessore 80286 dispone di alcune caratteristiche funzionali ad elevate prestazioni, adatte a supportare sistemi concorrenti multiutente: garantisce una protezione a livello hardware della memoria, mediante la separazione degli spazi di indirizzamento tra il sistema operativo e i programmi applicativi, e ha una velocità di esecuzione, alla frequenza di 10 MHz, sei volte maggiore di quella posseduta dall'8086, alla frequenza di 5 MHz. L'80286 è una estensione compatibile con l'8086/8088: in modalità a indirizzamento reale (*real address mode*), infatti, l'80286 può eseguire il codice oggetto dell'8086/8088, mentre in modalità a indirizzamento virtuale (*virtual address mode*) sono necessari alcuni adattamenti del codice sorgente dell'8086/8088, per sfruttare appieno il meccanismo di gestione virtuale della memoria, che viene offerto dall'architettura dell'80286 (Figura 3.1).

ARCHITETTURA DI BASE

L'80286 dispone di otto registri generali a 16 bit: AX, BX, CX, DX, SP, BP, SI e DI, come indicato in Figura 3.2. Ciascuno dei quattro registri AX, BX, CX e DX può essere utilizzato come registro di 16 bit oppure come insieme di due registri di 8 bit, per un totale di otto registri di 8 bit. I registri che terminano con la lettera "X" (ad esempio BX) utilizzano interamente i 16 bit a disposizione, mentre i due registri a 8 bit, interni ad un registro a 16 bit, vengono indicati con la notazione *Reg*L e *Reg*H (ad esempio AH e AL),

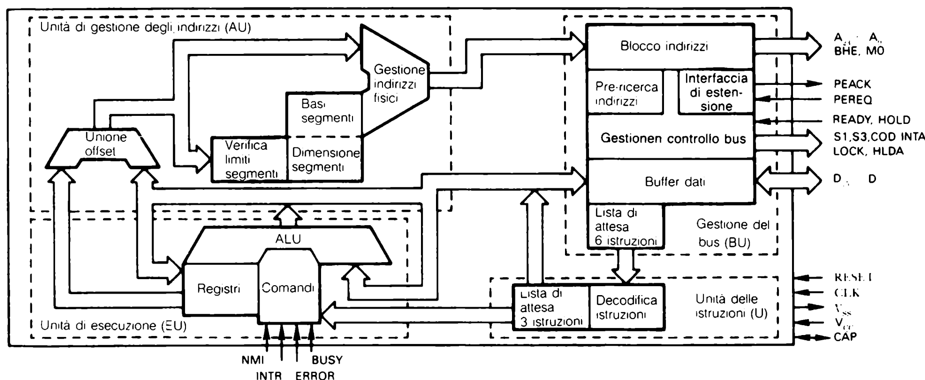


Figura 3.1 Architettura interna dell'80286

Bit 15 — Bit 0 (per *Reg*X)

Bit 7 — Bit 0 | Bit 7 — Bit 0 (per *Reg*H) (per *Reg*L)

AH	AL	AX	(Accumulatore)
BH	BL	BX	(Base)
CH	CL	CX	(Contatore)
DH	DL	DX	(Dati)

	SP	Puntatore di stack
	BP	Puntatore di base
	SI	Indice sorgente
	DI	Indice destinazione

Figura 3.2 Registri generali dell'80286

dove *Reg*L contiene gli otto bit meno significativi, mentre *Reg*H contiene gli otto bit più significativi.
I due registri a 16 bit SP (puntatore allo stack) e BP (puntatore alla base) vengono generalmente utilizzati nelle operazioni di gestione dello stack e contengono gli spiazzamenti delle locazioni che sono interne allo stack corrente. I due registri indice SI (indice sorgente) e DI (indice destinazione), referenziano inizialmente gli estremi di una zona di memoria, contenente ad esempio una struttura dati complessa, e vengono di solito incrementati o decrementati, per permettere un accesso completo a tale zona.

Registri di segmento
L'80286, tramite i registri di segmento a 16 bit CS, DS, SS ed ES, può referenziare contemporaneamente quattro segmenti distinti di memoria (Figura 3.3).
Il registro CS (segmento codice) indirizza la base del segmento di codice che è in esecuzione; il registro DS (segmento dati) indirizza la base del segmento di dati corrente, mentre il registro SS (segmento stack) indirizza la base del segmento di stack corrente (lo stack è una zona di memoria utilizzata per il passaggio dei parametri tra la procedura chiamata e il programma chiamante e in cui vengono allocati generalmente i risultati intermedi di una operazione). Inoltre, il programmatore ha accesso ad un secondo segmento di dati che viene indirizzato tramite il registro ES (segmento extra).
L'indirizzamento di una locazione di memoria, all'interno di un segmento, prevede innanzitutto la selezione del segmento, attraverso uno dei quattro registri di segmento (CS, DS, SS o ES) e, successivamente, l'uso di un offset di 16 bit, riferito alla base del segmento selezionato, che identifichi la locazione di memoria che interessa. In modalità di esecuzione a indirizzamento virtuale, il contenuto del registro di segmento e l'offset costituiscono, rispettivamente, i 16 bit più significativi e i 16 bit meno significativi di un indiriz-

Bit 15 — Bit 0

CS
DS
SS
ES

Figura 3.3 Registri di segmento dell'80286

zo virtuale (cioè logico) di 32 bit. Una volta che il segmento è stato selezionato, il programmatore può specificare, nell'istruzione, solo i 16 bit meno significativi dell'indirizzo virtuale, perché il microprocessore referenzi correttamente la locazione di memoria. In modalità di esecuzione a indirizzamento reale, il significato dei registri di segmento è lo stesso dei microprocessori 8086/8088.

Gli indirizzi dei segmenti di memoria vengono quindi interpretati dal microprocessore in maniera diversa, a seconda di quale modalità di esecuzione sia attiva. In modalità di indirizzamento reale, infatti, i registri di segmento contengono indirizzi fisici reali, mentre in modalità di indirizzamento virtuale i registri di segmento contengono indirizzi di memoria virtuali (indirizzi logici) che l'80286 converte automaticamente in indirizzi di memoria reali (indirizzi fisici).

Registri Indice, Puntatore e Base

Come abbiamo già sottolineato, l'indirizzo fisico di una locazione di memoria, all'interno di un segmento, si compone dell'indirizzo di base del segmento e di un offset, che può essere contenuto in uno dei registri puntatore, base o indice.

Le operazioni sullo stack utilizzano il selettore al segmento di stack (SS) e uno tra i due registri SP (puntatore allo stack) e BP (puntatore alla base). Gli offset, che referenziano una locazione di memoria all'interno dei segmenti dati DS ed ES, sono contenuti nel registro BX, mentre, per manipolare strutture dati più complesse, si utilizza, insieme al registro dati corrente, la coppia di registri indice SI (indice sorgente) e DI (indice destinazione).

Registri di stato e di controllo: i flag

L'80286 contiene un registro a 16 bit (Figura 3.4) suddiviso in undici campi o flag (alcuni bit di questo registro sono riservati o indefiniti).

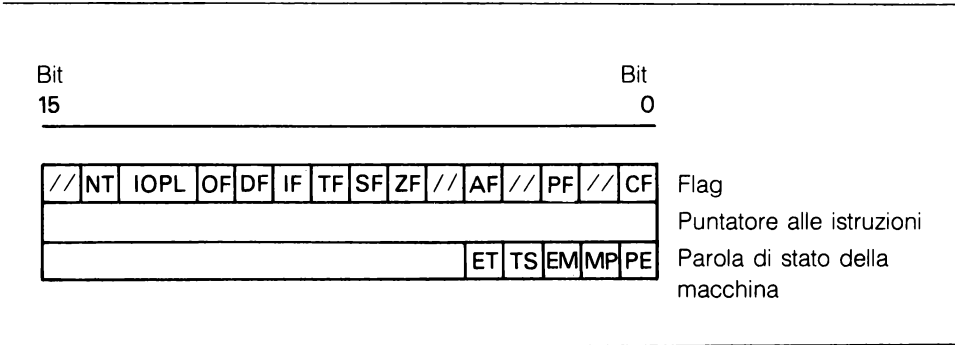


Figura 3.4 Registri di stato e di controllo dell'80286

Sei di questi flag, i flag di stato, vengono modificati da – e forniscono informazioni per – decisioni di controllo logiche e aritmetiche e sono i seguenti: CF (Carry), PF (Parità), AF (carry Ausiliario), ZF (Zero), SF (Segno) e OF (Overflow).

- Il flag Carry (CF) assume il valore 1 quando un'operazione aritmetica, eseguita su un operando di 8 o 16 bit, genera un riporto o chiede un prestito. In caso contrario, CF assume il valore 0. Questo flag viene utilizzato anche nelle istruzioni "shift" e "rotate" e contiene il bit che le istruzioni stesse hanno espulso dal registro.
- Il flag Parità (PF) viene utilizzato principalmente per scoprire eventuali errori di trasmissione di dati e assume il valore 1 nel caso di parità dispari e il valore 0 nel caso di parità pari.
- Il flag carry Ausiliario (AF) viene utilizzato nell'aritmetica BCD (Binary Coded Decimal: decimale codificata in binario) per indicare se si è verificato un riporto oppure un prestito nei 4 bit meno significativi di un valore numerico, espresso in notazione BCD.
- Il flag Zero (ZF) indica (valore 1) se il risultato dell'operazione è zero.
- Il flag Segno (SF) assume il valore 1 se il risultato dell'operazione è un numero negativo e assume il valore 0 se il risultato dell'operazione è un numero positivo (assumendo una rappresentazione dei numeri in complemento a due).
- Il flag Overflow (OF) indica il superamento dei limiti, per operazioni in aritmetica con segno.

Tre degli undici flag – TF, IF e DF – abilitano alcune funzioni speciali del microprocessore: con il flag Trappola (TF) a 1, viene generata un'interruzione ad ogni istruzione; questa modalità di esecuzione passo-passo può essere molto utile in fase di collaudo dei programmi; il flag di Interrupt (IF) abilita (valore 1) o disabilita (valore 0) le interruzioni esterne, mentre il flag Direzione (DF) precisa in quale direzione della memoria (verso indirizzi crescenti o decrescenti) hanno luogo le operazioni sulle stringhe di caratteri (se DF è a 0, SI e/o DI si autoincrementano; se DF è a 1, SI e/o DI si autodecrementano).
I flag IOPL (Input/Output Privilege Level) e NT (Nested Task) sono significativi solo quando l'80286 esegue in modalità virtuale protetta (più precisamente, il termine *virtuale* specifica il tipo di indirizzamento, mentre il termine *protetta* indica l'esistenza di un meccanismo di protezione a livello hardware che isola i processi e, all'interno di un processo, i diversi strati di software. In seguito, parlando di modalità di esecuzione, utilizzeremo uno dei due termini o entrambi a seconda del contesto in cui ci troveremo). Il flag IOPL si compone di due bit e definisce il massimo livello di privilegio con cui il processo in esecuzione può eseguire istruzioni di ingresso e uscita, mentre il flag NT indica se il processo corrente è annidato a livello hardware in un

altro processo, cioè se il processo in esecuzione dispone di un puntatore valido ad un processo concatenato (NT = 1) oppure no (NT = 0).

Program counter
Il registro IP (puntatore all'istruzione corrente), indicato in Figura 3.4, contiene l'offset (16 bit) che indirizza la prossima istruzione che il microprocessore deve eseguire, all'interno del segmento di codice corrente, identificato dal contenuto del registro CS (16 bit).

Parola di stato dell'80286
I primi cinque bit meno significativi della parola di stato di 16 bit dell'80286 (Figura 3.4) contengono, rispettivamente, il bit Protezione (PE: Protection Enable), il bit di presenza del coprocessore matematico (MP: Monitor Processor Extension), il bit di emulazione delle funzioni del coprocessore matematico (EM: Processor Extension Emulated), il bit di cambio contesto di processo (TS: Task Switch) e il bit indicatore del tipo di estensione del processore (ET: Processor Extension Type).

- Il bit di protezione (PE) viene utilizzato per far funzionare il microprocessore in modalità a indirizzamento protetta (PE a 1), oppure in modalità a indirizzamento reale (PE a 0).
- Il bit di presenza del coprocessore matematico (MP) indica alla CPU se è attivato anche il coprocessore 80287 (80387) e controlla la funzione dell'istruzione WAIT. Quando viene eseguita questa istruzione, se la CPU trova il bit MP a 1, allora controlla il bit TS. Se anche il bit TS è a 1, la CPU genera un'eccezione con cui indica la non disponibilità del coprocessore.
- Il bit di emulazione delle funzioni del coprocessore (EM) indica alla CPU che le funzioni dell'80287 (80387) devono essere emulate da software (EM a 1).
- Il bit di task-switch (TS) viene posto a 1 dalla CPU quando il controllo dell'unità centrale passa in modo sincrono da un processo ad un altro. Se TS è attivo, un'istruzione per il coprocessore genera un'eccezione di "coprocessore non disponibile".

3.2 Il microprocessore 80386

L'80386 è un microprocessore a 32 bit, particolarmente adatto a supportare i sistemi operativi che sono destinati ad operare in ambiente multiprogrammato. Con i suoi registri a 32 bit, l'80386 è in grado di gestire indirizzi e tipi di dati di 32 bit.
Questo microprocessore può indirizzare fino a quattro gigabyte di memoria fisica e 64 terabyte (2^{46} byte) di memoria virtuale, integra sul chip un mec-

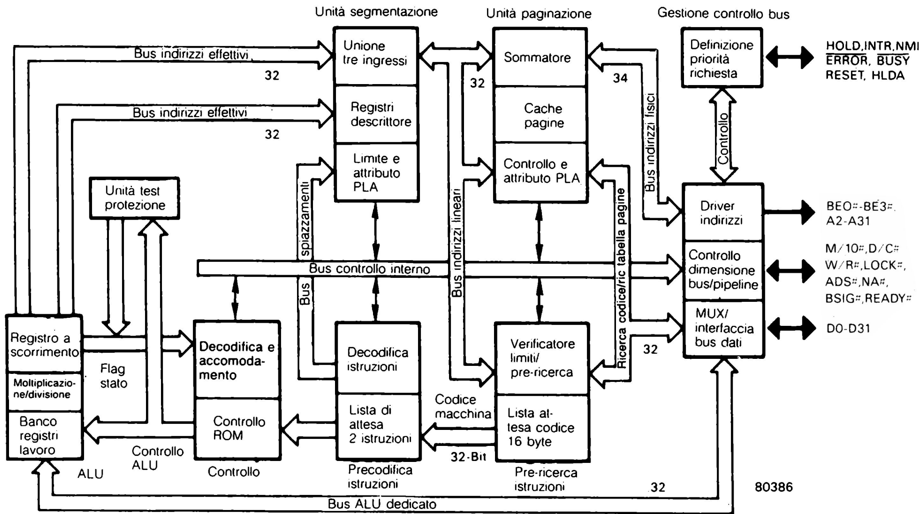


Figura 3.5 Meccanismo di pipelining dell'80386

canismo molto efficiente di protezione e di gestione della memoria e traduce a livello hardware gli indirizzi virtuali in indirizzi fisici.
Un meccanismo di pipelining (Figura 3.5) – cioè la presenza di più stadi di esecuzione dell'istruzione gestiti in parallelo da unità dedicate –, la più estesa banda passante del bus e la traduzione a livello hardware degli indirizzi permettono all'80386 di ridurre considerevolmente i tempi di esecuzione, raggiungendo velocità di 3-4 milioni di istruzioni al secondo.
L'80386 dispone anche di meccanismi di autodiagnosi, della possibilità di accedere da programma alla memoria cache interna (usata per la traduzione da indirizzi logici a indirizzi fisici), di quattro registri di breakpoint. Costituisce inoltre una ulteriore estensione compatibile con i microprocessori 8086/8088/80286.

TIPI DI DATI

Il microprocessore 80386 supporta diversi tipi di dati, in aggiunta a quelli che il programmatore può definire con l'8086/80286 (ad esempio gli interi a 32 bit con o senza segno e i dati con un formato variabile tra 1 e 32 bit) e, oltre ai tipi puntatore standard definiti per la famiglia 8086/80286, supporta anche puntatori di 48 bit, con offset di 32 bit.

INDIRIZZAMENTO DEGLI OPERANDI

In aggiunta agli operandi immediati di 8 e 16 bit, supportati dai microprocessori 8086/8088/80286, l'80386 supporta anche operandi di 32 bit. In caso di indirizzamento immediato a 16 bit, l'operando viene automaticamente esteso a 32 bit dal microprocessore.

Calcolo dell'indirizzo effettivo

L'accesso ad una locazione di memoria interna ad un segmento di dimensione maggiore di 64 kB può essere ottenuto con un indirizzo effettivo di 32 bit o di 16 bit. Questo indirizzo è costituito dal contenuto di un registro base (opzionale), di un registro indice (opzionale) e di uno spiazzamento (opzionale). Inoltre, i modi di indirizzamento a 32 bit sono stati estesi, consentendo a qualunque registro generale di essere utilizzato con le funzioni di un registro base o di un registro indice.

ESECUZIONE DEL CODICE 8086

L'80386 può operare in due modalità di indirizzamento che sono compatibili con il set di istruzioni dell'8086/80286 e che permettono di eseguire il codice oggetto dell'8086. La modalità reale è identica a quella dell'80286 e costituisce la modalità attiva nel microprocessore, dopo un'operazione di RESET. La cosiddetta "modalità 8086 virtuale" presenta delle caratteristiche comuni rispetto a quella dell'80386 e permette al codice dell'8086 di essere eseguito sull'80386 in modalità protetta, con una gestione della memoria a pagine.

ARCHITETTURA DI BASE

Il microprocessore 80386 dispone di 32 registri, che possono essere suddivisi in sette categorie fondamentali:

- Registri generali
- Registri di segmento
- Flag e program counter
- Registri di controllo
- Registri di indirizzamento sistema
- Registri di debug e di ricerca

Questi registri sono un'estensione dei registri a 16 bit di cui dispongono l'8086 e l'80286.

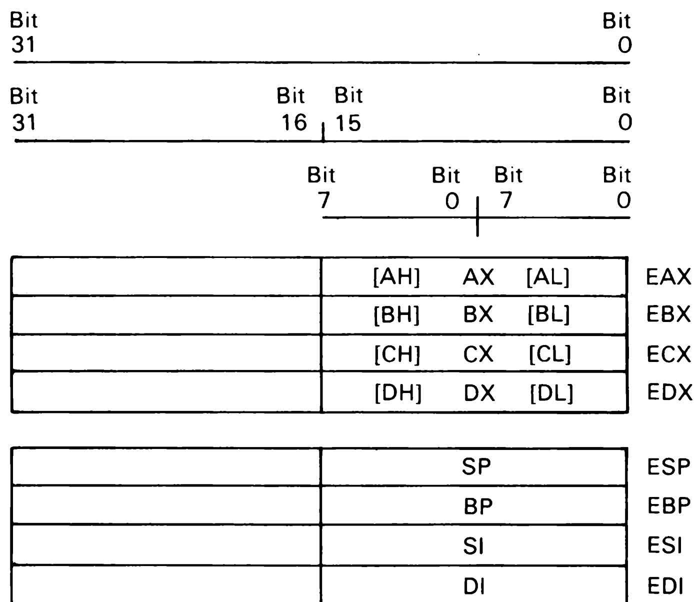


Figura 3.6 Registri generali dell'80386

Registri generali

L'80386 dispone di otto registri generali (Figura 3.6) che svolgono le stesse funzioni realizzate dagli otto registri generali dell'80286, con la sola eccezione che i registri dell'80386 hanno una dimensione di 32 bit (per avere maggiori informazioni sul significato e l'utilizzo di questi registri, rileggete il paragrafo che è stato dedicato, in questo capitolo, alla descrizione dei registri generali del microprocessore 80286).

I registri generali possono supportare dati di 1, 8, 16 e 32 bit, dati con una dimensione in bit variabile tra 1 e 32 e indirizzi di 16 e 32 bit. Gli otto registri sono: EAX (accumulatore), EBX (base), EDX (dati), ESP (puntatore allo stack), EBP (puntatore alla base), ESI (indice sorgente) e EDI (indice destinazione).

Per accedere a tutti i 32 bit di un registro, gli identificatori di registro devono iniziare con la lettera "E", mentre l'assenza di questo prefisso dimezza automaticamente la capacità di memorizzazione dei registri, che diventano così equivalenti ai corrispondenti registri dell'8086/80286.

Registri di segmento

Il microprocessore 80386 dispone di sei registri di segmento a 16 bit (Figura 3.7), che contengono i selettori ad altrettante aree di memoria (segmenti) che

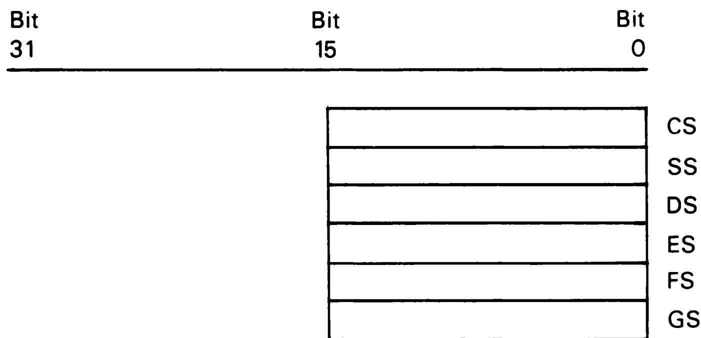


Figura 3.7 Registri di segmento dell'80386

sono correntemente indirizzabili. In modalità di indirizzamento reale, un segmento può assumere una dimensione variabile tra 1 byte e 64 kB (2^{16} byte), mentre in modalità di indirizzamento virtuale protetta, la dimensione di un segmento può variare tra 1 byte e 4 gigabyte (2^{32} byte).
I sei registri di segmento dell'80386 sono: CS (segmento codice), DS (segmento dati), SS (segmento stack), ES (segmento dati extra), FS e GS. La funzione generalmente svolta dai registri FS e GS è quella di consentire un uso più flessibile delle istruzioni di accesso a strutture dati caratterizzate da base e indice, che nei precedenti microprocessori utilizzavano praticamente solo il registro di segmento ES.

Program counter e registro dei flag
Il microprocessore 80386 dispone di un registro a 32 bit (EIP) che contiene il puntatore all'istruzione corrente (Figura 3.8), cioè lo spiazzamento (offset)

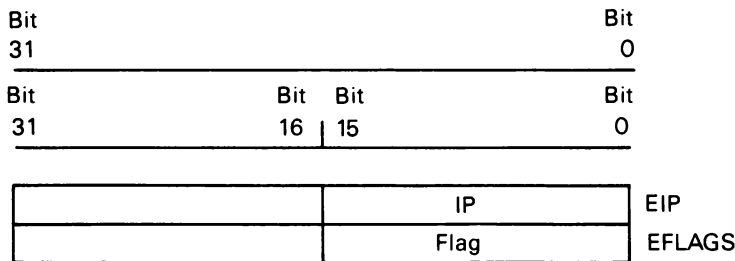


Figura 3.8 Registri EIP e EFLAGS dell'80386

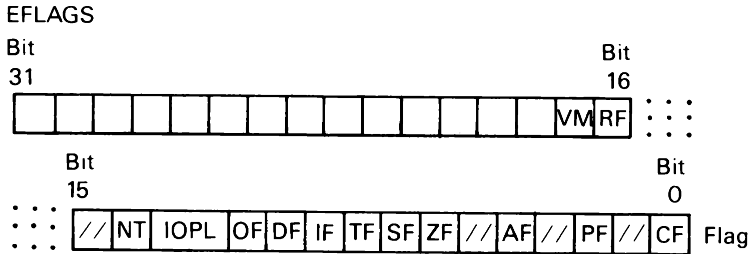


Figura 3.9 Struttura interna del registro EFLAGS

dell'istruzione corrente rispetto alla base del segmento di codice CS. Il programmatore, nel caso di indirizzamenti di 16 bit, utilizza solo i sedici bit meno significativi del registro EIP, come se si trattasse di un normale registro di 16 bit (registro IP).

Il registro dei flag (EFLAGS) è anch'esso a 32 bit (Figura 3.9) e la sua funzione è quella di regolare lo svolgimento di alcune operazioni e di indicare lo stato del microprocessore. Questo registro contiene due nuovi flag, in aggiunta a quelli già definiti sull'80286: VM (flag di modalità 8086 virtuale) e RF (flag di ripristino del controllo di correttezza: Resume). I 16 bit meno significativi del registro dei flag dell'80386 contengono gli stessi flag di controllo e di stato che sono disponibili sui microprocessori 8086/80286 (per avere maggiori informazioni sul significato di questi flag, rileggete il paragrafo di questo capitolo dedicato ai flag del microprocessore 80286).

Il flag VM abilita, sull'80386, la modalità a indirizzamento 8086 virtuale. Infatti, se VM è a 1, e l'80386 si trova in modalità protetta, il microprocessore passa in modalità virtuale 8086 e simula così il funzionamento dell'8086 (anche per quanto riguarda la gestione dei registri di segmento), generando l'eccezione numero 13, nel caso di esecuzione di qualche istruzione privilegiata dell'80386.

Il flag RF abilita (RF a 0) o disabilita (RF a 1) il controllo sulla correttezza dell'istruzione successiva a quella corrente e viene automaticamente posta a 0 dopo la corretta terminazione dell'istruzione corrente. Viene utilizzata insieme ai registri di breakpoint nelle fasi di collaudo dei programmi.

Registri di controllo

Il microprocessore 80386 dispone di tre registri di controllo a 32 bit: CR0, CR2 e CR3 (Figura 3.10). Questi registri contengono informazioni sullo stato della macchina, indipendenti dal processo in esecuzione e accessibili con istruzioni "load" e "store".

Bit 31	Bit 24 Bit 23	Bit 16	Bit 15	Bit 8 Bit 7	Bit 0	
			Parola di stato macchina / |ET|TS|EM|MP|PE|			CR0
						CR2
						CR3

Figura 3.10 Registri di controllo dell'80386

Il registro CR0 contiene sei flag, che sono utilizzati dal microprocessore con funzioni di controllo e come indicatori di stato. I 16 bit meno significativi del registro CR0 costituiscono la parola di stato (MSW: Machine Status Word), per cui, in modalità protetta, l'80386 è compatibile con l'80286. I comandi LMSW e SMSW possono indifferentemente essere utilizzati sull'80386 e sull'80286, mentre se il programmatore ha intenzione di accedere all'intero registro CR0, deve servirsi dell'istruzione MOV CR0,*Registro*.
Il significato dei bit ET (tipo di estensione del processore), TS (task-switch), EM (simulazione coprocessore), MP (presenza coprocessore) e PE (modalità protetta) rimane identico a quello già illustrato per il microprocessore 80286.
Il registro CR1 è riservato dalla Intel per i microprocessori di prossima realizzazione, mentre il registro CR2 contiene l'indirizzo di 32 bit che ha causato l'ultima richiesta di allocazione di una pagina in memoria. Il registro CR3 memorizza l'indirizzo fisico di base della tabella contenente il direttorio delle pagine di memoria (*page table directory*). Tale indirizzo è sempre quello iniziale di una pagina (dedicata appunto alla tabella del direttorio), quindi i suoi 12 bit meno significativi valgono sempre zero.

Registri di sistema
Il microprocessore 80386 (come l'80286) dispone di quattro registri di sistema – GDTR, IDTR, LDTR e TR – (Figura 3.11), con cui indirizza, in modalità protetta, alcune tabelle (GDT o *global descriptor table*, IDT o *interrupt descriptor table* e LDT o *local descriptor table*), contenenti le informazioni sui segmenti allocati in memoria, e un segmento (TSS o *task state segment*), contenente le informazioni sullo stato del processo in esecuzione.
GDTR e IDTR sono registri a 48 bit e contengono i 32 bit dell'indirizzo di base e i 16 bit della dimensione della GDT e della IDT. LDTR e TR sono registri a 16 bit e contengono il selettore di 16 bit ai segmenti LDT e TSS del processo corrente.

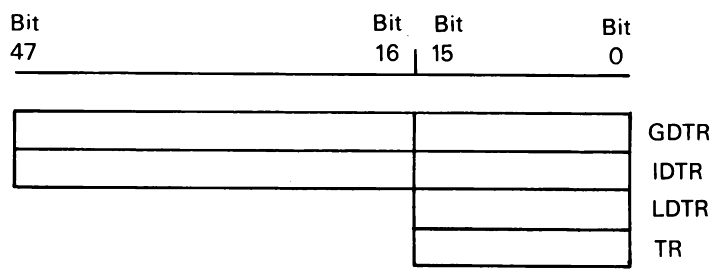


Figura 3.11 Registri di indirizzamento di sistema

Registri di debug e registri di ricerca

Il microprocessore dispone di sei registri di debug a 32 bit: DR0, DR1, DR2, DR3, DR6 e DR7 (Figura 3.12). DR4 e DR5 sono riservati alla Intel; DR6 definisce una condizione di breakpoint, mentre DR7 visualizza lo stato corrente dei breakpoint.

Il microprocessore 80386 dispone anche di due registri di "ricerca" a 32 bit (Figura 3.13) usati per verificare il contenuto della RAM e della CAM (Content Addressable Memory: memoria associativa) presenti nella cache utilizzata nella traduzione da indirizzi logici a indirizzi fisici.

TR6 serve per definire il criterio di ricerca di informazioni nella cache, mentre TR7 riporta il risultato di una ricerca.

Bit 31 — Bit 0	
Indirizzo lineare breakpoint 0	DR0
Indirizzo lineare breakpoint 1	DR1
Indirizzo lineare breakpoint 2	DR2
Indirizzo lineare breakpoint 3	DR3
Riservato Intel	DR4
Riservato Intel	DR5
Stato breakpoint	DR6
Controllo breakpoint	DR7

Figura 3.12 Registri di debug dell'80386

Controllo ricerca	TR6
Stato ricerca	TR7

Figura 3.13 Registri di ricerca dell'80386

3.3 Set di istruzioni dell'80286/80386

AAA

(Ascii Adjust al after Addition)

Durata (cicli): 3 (80286), 4 (80386)

Descrizione: Sistemazione decimale del risultato di un'addizione

Operazione: L'istruzione AAA deve essere eseguita solo dopo un'istruzione ADD, che genera come risultato un byte e lo memorizza nel registro AL. AAA converte il contenuto di AL in una cifra decimale in codice BCD in formato non compattato (cioè una cifra BCD ogni byte).
AAA esamina i quattro bit meno significativi di AL per verificare se contengono un numero valido nel formato BCD (decimale codificato in binario), di valore compreso tra 0 e 9; inoltre, azzera i quattro bit più significativi di AL e i flag AF (Carry ausiliario) e CF (Carry), se era stato generato un riporto decimale. Se il valore contenuto nei quattro bit meno significativi di AL è maggiore di 9, o se il flag AF è a 1, AAA esegue le seguenti azioni: incrementa di 6 il registro AL, incrementa AH di uno, pone a 1 i flag AF e CF, e azzera i quattro bit più significativi di AL.

Sintassi: AAA (nessun operando)

Flag modificati: AF, CF

Flag indefiniti: OF, ZF, SF, PF

Eccezioni in modalità protetta: Nessuna

Eccezioni in modalità reale: Nessuna

Nota per l'80386: Nessuna eccezione

Esempio:

```
ADD    AL,BL    ;somma i numeri BCD in AL e BL
AAA             ;restituisce un risultato BCD nel formato non impaccato
```

AAD
(Ascii Adjust ax before Division)

Durata (cicli): 14 (80286), 19 (80386)

Descrizione: Sistemazione decimale prima di una divisione

Operazione: L'istruzione AAD prepara due cifre BCD, espresse nel formato non compattato (la cifra meno significativa è memorizzata nel registro AL, mentre la cifra più significativa nel registro AH), ad una operazione di divisione che restituisce un risultato nel formato non compattato. Il registro AL viene caricato con il valore AL+(10 × AH) e il registro AH viene azzerato. In questo modo, il registro AX contiene il valore binario equivalente al numero di partenza di due cifre nel formato non compattato.

Sintassi: AAD (nessun operando)

Flag modificati: SF, ZF, PF

Flag indefiniti: OF, AF, CF

Eccezioni in modalità protetta: Nessuna

Eccezioni in modalità reale: Nessuna

Nota per l'80386: Nessuna eccezione

Esempio:

```
AAD                    ;sempre prima dell'istruzione di divisione
```

AAM
(Ascii Adjust ax after Multiply)

Durata (cicli): 16 (80286), 17 (80386)

Descrizione: Sistemazione decimale del risultato di una moltiplicazione

Operazione: L'istruzione AAM deve essere eseguita solo dopo l'istruzione MUL che opera su due cifre BCD nel formato non compattato, memorizzando il risultato nel registro AX. Poiché questo risultato è minore di 100, è contenuto interamente nel registro AL.
AAM trasforma nel formato non compattato il risultato contenuto in AL, dividendolo per 10 e memorizzando la cifra più significativa (quoziente) in AL e la cifra meno significativa (resto) nel registro AH.

Sintassi: AAM (nessun operando)

Flag modificati: SF, ZF, PF

Flag indefiniti: OF, AF, CF

Eccezioni in modalità protetta: Nessuna

Eccezioni in modalità reale: Nessuna

Nota per l'80386: Nessuna eccezione

Esempio:
```
AAM                    ;sempre dopo l'istruzione di moltiplicazione
```

AAS
(Ascii Adjust al after Subtraction)

Durata (cicli): 3 (80286), 4 (80386)

Descrizione: Sistemazione decimale del risultato di una sottrazione

Operazione: L'istruzione AAS deve essere eseguita solo dopo un'istruzione di sottrazione, che genera un risultato di un byte e lo memorizza nel registro AL.
L'istruzione AAS memorizza nel registro AL il risultato dell'operazione nel formato decimale non compattato, nel rispetto delle seguenti regole: se i quattro bit meno significativi del registro AL sono maggiori di 9, oppure se il flag AF è a 1, il registro AL viene decrementato di 6, il registro AH viene decrementato di 1 e i flag CF e AF assumono il valore 1. In caso contrario, i flag CF e AF vengono azzerati.
Il contenuto originale del registro AL viene quindi sostituito con un byte, i cui quattro bit più significativi sono tutti a zero, mentre i quattro bit meno significativi rappresentano un numero compreso tra 0 e 9.

Sintassi: AAS (nessun operando)

Flag modificati: AF, CF

Flag indefiniti: OF, SF, ZF, PF

Eccezioni in modalità protetta: Nessuna

Eccezioni in modalità reale: Nessuna

Nota per l'80386: Nessuna eccezione

Esempio:
```
ASS                    ;sempre dopo l'istruzione di sottrazione
```

ADC
(ADd with Carry)

Durata (cicli): 2 – 7 (80286), 2 – 7 (80386)

Descrizione: Somma due operandi con riporto

Operazione: L'istruzione ADC esegue la somma intera di due operandi. Se il flag CF è a 1, viene aggiunta una unità alla somma dei due operandi e il risultato viene memorizzato nell'operando destinazione.

Sintassi: ADC *destinazione,sorgente*

Flag modificati: OF, SF, ZF, AF, PF, CF

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Se si cerca di memorizzare il risultato in un segmento in cui non è permessa la scrittura, il microprocessore genera una eccezione generale di protezione.
Si genera una eccezione generale di protezione anche nel caso di caricamento nei registri CS, DS o ES di un indirizzo effettivo di memoria non lecito, mentre se si tenta di caricare il registro SS con un indirizzo illegale, viene generata una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 nel caso di violazione dei limiti di segmento.

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:
Somma con riporto di due operandi immediati e il risultato in accumulatore:

```
ADC     AL,4
ADC     AX,298
ADC     EBX,22334455H       ;solo 80386
```

Somma con riporto di due operandi immediati e risultato in un registro o in una locazione di memoria:

```
ADC     CX,341
ADC     BL,10
ADC     TABELLA[SI],2
ADC     MEMORIA,6293
ADC     NUMERO,12345678     ;solo 80386
```

Somma con riporto di dati da: Registro a registro
Registro a memoria
Memoria a registro

```
ADC     DL,BL
ADC     MEM_WRD,AX
ADC     SI,MEM_WRD
```

ADD
(ADDition)

Durata (cicli): 2 – 7 (80286), 2 – 7 (80386)

Descrizione: Somma due operandi

Operazione: L'istruzione ADD esegue la somma di due operandi e memorizza il risultato nell'operando destinazione.

Sintassi: ADD *destinazione,sorgente*

Flag modificati: AF, CF, OF, PF, SF, ZF

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Se si cerca di memorizzare il risultato in un segmento in cui non è permessa la scrittura, il microprocessore genera una eccezione generale di protezione. Se i registri di segmento CS, DS e ES contengono un indirizzo effettivo di memoria non lecito, si genera ugualmente una eccezione generale di protezione, mentre se il registro di segmento SS contiene un indirizzo illegale, viene generata una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 nel caso di violazione dei limiti di segmento.

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:
Da registro e registro a registro:
```
ADD     AX,BX
ADD     EBX,ECX             ;solo 80386
```
Da registro e memoria a memoria:
```
ADD     EXTMEM,AX
```
Da memoria e registro a registro:
```
ADD     DX,BUFF
```
Da operando immediato e accumulatore ad accumulatore:
```
ADD     AL,4
ADD     EAX,98765432H       ;solo 80386
```
Da operando immediato e registro a registro:
```
ADD     CX,1985
```
Da operando immediato e memoria a memoria:
```
ADD     EXTMEM,23
```

AND
(logical AND)

Durata (cicli): 2 – 7 (80286), 2 – 7 (80386)

Descrizione: AND logico di due operandi

Operazione: Il risultato contiene un bit a 1 in quelle posizioni di bit in cui entrambi gli operandi contengono un bit a 1; in tutti gli altri casi, i bit del risultato sono a 0. Inoltre, vengono azzerati i flag OF e CF.

Sintassi: AND *destinazione,sorgente*

Flag modificati: OF = 0, CF = 0, PF, SF, ZF

Flag indefiniti: AF

Eccezioni in modalità protetta: Se si cerca di memorizzare il risultato in un segmento in cui non è permessa la scrittura, il microprocessore genera una eccezione generale di protezione.
Si genera una eccezione generale di protezione anche nel caso di tentativo di caricamento di un indirizzo effettivo di memoria non lecito nei registri CS, DS o ES, mentre se si cerca di caricare il registro SS con un indirizzo illegale, viene generata una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 nel caso di violazione dei limiti di segmento.

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:
Da operando immediato e registro a registro:
```
AND     BL,11001110B
AND     EAX,11110000111100001111000011110000B     ;solo 80386
```
Da operando immediato e memoria a memoria:
```
AND     EXTMEM,10011100B
AND     NUMERO,0F0F0F0F0H        ;solo 80386
```
Da operando immediato e accumulatore ad accumulatore:
```
AND     AX,1001111101010110B
AND     AL,N_BYTE
```
Da registro e registro a registro:
```
AND     AX,BX
AND     EAX,ECX  ;solo 80386
```
Da registro e memoria a memoria:
```
AND     EXTMEM,SI
```
Da memoria e registro a registro:
```
AND     DH,EXTBYTE
```

ARPL

(Adjust RPL field of selector)

Durata (cicli): 10 – 11 (80286), 20 – 21 (80386)

Descrizione: Modifica il contenuto del campo RPL del selettore

Operazione: L'istruzione privilegiata ARPL serve al sistema operativo per evitare che si verifichi un accesso illecito ad un segmento protetto da parte di una procedura chiamante che non possiede i diritti richiesti (anche se la chiamata arriva da un'altra procedura del sistema operativo).
Il primo dei due operandi di ARPL è una variabile di memoria di 16 bit, o un registro, che contiene il valore di un selettore, mentre il secondo operando è un registro. Se il campo RPL (2 bit) del primo operando ha un valore inferiore rispetto al campo RPL del secondo operando, il microprocessore pone a 1 il flag Zero e incrementa il campo RPL del primo operando fino a eguagliare il campo RPL del secondo operando. In caso contrario, il flag Zero viene azzerato e nessun cambiamento viene eseguito sul primo operando.

Sintassi: ARPL (*selettore,selettore_CS*)

Flag modificati: ZF

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Il microprocessore genera una eccezione generale di protezione se si cerca di memorizzare il risultato in un segmento in cui non è permessa la scrittura oppure se i registri di segmento CS, DS ed ES contengono un indirizzo effettivo di memoria non lecito, mentre se il registro di segmento SS contiene un indirizzo illegale, viene generata una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 6. L'istruzione ARPL, infatti, non è riconosciuta in modalità di indirizzamento reale.

Nota per l'80386: Nessuna eccezione

Esempio:

```
ARPL    MEM_WRD,BX
```

BOUND
(check array index against BOUNDs)

Durata (cicli): 13 (80286), 10 (80386)

Descrizione: Si genera l'interruzione INT 5 se il contenuto del registro non è interno ai limiti prefissati

Operazione: L'istruzione BOUND verifica se un numero con segno – che rappresenta un indice di un array – cade nei limiti definiti da due parole di memoria. Il primo operando (un registro) deve essere maggiore o uguale al valore contenuto nella prima parola di memoria e minore o uguale al valore contenuto nella seconda parola di memoria. Se queste condizioni non sono soddisfatte, si verifica l'interruzione INT 5.

Sintassi: BOUND *destinazione,sorgente*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata l'interruzione INT 5 se la verifica sui limiti di campo fornisce un esito negativo; il microprocessore genera, invece, una eccezione generale di protezione nel caso di tentativo di caricamento di un indirizzo effettivo di memoria non lecito nei registri CS, DS o ES, mentre se si cerca di caricare il registro SS con un indirizzo illegale, viene generata una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 5, se la verifica sui limiti di campo fornisce un esito negativo, e l'interruzione INT 13, se il secondo operando ha un offset di 0FFFDH o maggiore. Se, invece, il secondo operando è un registro, viene generata l'interruzione 6.

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:
```
BOUND   AX,MEM_WORD
BOUND   EBX,NEW_WORD      ;solo 80386
```

CALL
(CALL procedure)

Durata (cicli): 7 – 185 (80286), 3 – 275 (80386)

Descrizione: Salva l'indirizzo della prossima istruzione sullo stack, trasferisce il controllo del programma

Operazione: L'istruzione CALL memorizza sullo stack l'indirizzo di ritorno, cioè l'indirizzo dell'istruzione successiva a quella di chiamata e trasferisce il controllo del programma alla locazione di memoria referenziata. Quando la procedura chiamata ha completato la sua esecuzione, il programma chiamante riprende ad eseguire a partire dall'istruzione memorizzata sullo stack.

Sintassi: CALL *operando*

Flag modificati: Nessuno (tranne nel caso di un task-switch, cioè di un cambio di contesto di processo).

Flag indefiniti: Nessuno

Eccezioni in modalità protetta:
- CALL di tipo FAR: possono essere generate l'eccezione generale di protezione, l'eccezione di assenza del descrittore, l'eccezione di stack e l'eccezione di segmento di stato di processo non valido.
- CALL dirette di tipo NEAR: può essere generata l'eccezione generale di protezione, se la locazione referenziata della procedura viola i limiti del segmento di codice.
- CALL indirette di tipo NEAR: viene generata l'eccezione generale di protezione, se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito oppure se l'offset della locazione referenziata viola i limiti del segmento di codice; il microprocessore genera anche l'eccezione di stack, se il registro di segmento SS contiene un indirizzo non legale.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 nel caso di violazione dei limiti di segmento.

Nota per l'80386: Gli operandi di tipo puntatore sono di 48 bit.

Esempio:
```
CALL    SOMMA   ;procedura interna o esterna al segmento di codice
```

Solo 80386:
```
CALL    PUNT                  ;gli operandi di tipo puntatore sono di 48 bit
CALL    SPIAZZ                ;lo spiazzamento è di 32 bit
CALL    registro o memoria    ;gli operandi di tipo word sono di 32 bit
```

CBW

(Convert Byte into Word)

Durata (cicli): 2 (80286), 3 (80386)

Descrizione: Converte un byte in una word (AH = bit più significativo di AL)

Operazione: L'istruzione CBW converte un byte con segno, contenuto in AL, in una word con segno e la memorizza in AX: questa operazione viene realizzata estendendo il bit più significativo di AL in tutti i bit di AH.

Sintassi: CBW (nessun operando)

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Nessuna

Eccezioni in modalità reale: Nessuna

Nota per l'80386: Vale quanto detto per l'80286. Inoltre, esiste l'istruzione CWDE che estende un operando di tipo word, memorizzato in AX, in una doubleword memorizzata in EAX.

Esempio:

```
CBW
```

CLC
(CLear Carry flag)

Durata (cicli): 2

Descrizione: Flag Carry a 0

Operazione: L'istruzione CLC azzera il flag Carry, senza modificare altri registri o flag.

Sintassi: CLC (nessun operando)

Flag modificati: CF = 0

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Nessuna

Eccezioni in modalità reale: Nessuna

Nota per l'80386: Nessuna eccezione

Esempio:

```
CLC
```

CLD
(CLear Direction flag)

Durata (cicli): 2

Descrizione: Flag Direzione a 0, autoincremento di SI e DI

Operazione: L'istruzione CLD azzera il flag Direzione, senza modificare altri registri o flag. Dopo che CLD è stata eseguita, le operazioni sulle stringhe incrementano automaticamente i registri indice (SI e/o DI).

Sintassi: CLD (nessun operando)

Flag modificati: DF

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Nessuna

Eccezioni in modalità reale: Nessuna

Nota per l'80386: Nessuna eccezione

Esempio:

```
CLD
```

CLI
(CLear Interrupt flag)

Durata (cicli): 3

Descrizione: Disabilitazione globale delle interruzioni (flag Interrupt a 0)

Operazione: L'istruzione CLI azzera il flag IF, senza modificare altri flag. Questa operazione ha l'effetto di disabilitare tutte le interruzioni, ad eccezione di quelle non mascherabili, che si presentano sulla linea NMI.

Sintassi: CLI (nessun operando)

Flag modificati: IF = 0

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Quando il livello di privilegio corrente (CPL) è maggiore del contenuto dell'indicatore IOPL (livello di privilegio per operazioni di Input/Output), viene generata una eccezione generale di protezione. IOPL indica il livello di privilegio minimo che deve possedere il processo per eseguire le operazioni di I/O.

Eccezioni in modalità reale: Nessuna

Nota per l'80386: Nessuna eccezione

Esempio:
```
CLI
```

CLTS
(CLear Task Switched flag)

Durata (cicli): 2 (80286), 5 (80386)

Descrizione: Flag di Task-Switch a 0

Operazione: L'istruzione CLTS azzera il flag di Task-Switch (TS), nella parola di stato del microprocessore. CLTS è un'istruzione privilegiata che può essere eseguita solo con livello di privilegio 0 ed è riservata al sistema operativo. Il flag TS viene utilizzato per gestire il coprocessore matematico, nel modo seguente: ogni esecuzione di una istruzione WAIT o ESC genera l'interruzione INT 7, che indica la non disponibilità del coprocessore matematico, se i flag MP e TS sono entrambi a 1.

Sintassi: CLTS (nessun operando)

Flag modificati: TS = 0

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Quando il livello di privilegio corrente (CPL) contiene un valore diverso da 0, il microprocessore genera una eccezione generale di protezione, se viene eseguita l'istruzione CLTS.

Eccezioni in modalità reale: Nessuna

Nota per l'80386: Nessuna eccezione

Esempio:
```
CLTS
```

CMC
(CoMplement Carry flag)

Durata (cicli): 2

Descrizione: Complementa il flag Carry

Operazione: L'istruzione CMC complementa il bit del flag Carry, senza modificare altri flag. Se il flag Carry è a 1, viene azzerato, mentre se è a 0, assume il valore 1.

Sintassi: CMC (nessun operando)

Flag modificati: CF

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Nessuna

Eccezioni in modalità reale: Nessuna

Nota per l'80386: Nessuna eccezione

Esempio:

```
CMC
```

CMP
(CoMPare two operands)

Durata (cicli): 2 – 7 (80286), 2 – 6 (80386)

Descrizione: Sottrazione degli operandi, con modifica dei flag

Operazione: L'istruzione CMP sottrae l'operando sorgente dall'operando destinazione, modificando alcuni flag e non alterando i valori dei due operandi, che devono essere dello stesso tipo, tranne quando CMP opera in modalità di indirizzamento immediato, in cui un byte di dato immediato può essere confrontato con una word di memoria.

Sintassi: CMP *destinazione,sorgente*

Flag modificati: OF, SF, ZF, AF, PF, CF

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, viene generata una eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, viene generata una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13, nel caso di violazione dei limiti di segmento.

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:
Operando immediato con memoria:
```
CMP     EXTMEM,5CAFH
```
Operando immediato con registro:
```
CMP     BL,5
CMP     EAX,0FFFF0000H      ;solo 80386
```
Operando immediato con accumulatore:
```
CMP     AL,7
```
Registro con registro:
```
CMP     AX,BX
CMP     EDX,EAX  ;solo 80386
```
Registro con memoria:
```
CMP     EXTMEM,DX
```
Memoria con registro:
```
CMP     CH,EXTMEM
CMP     EBX,VALORE     ;solo 80386
```

CMPS / CMPSB / CMPSW
(CoMPare String/String Byte/String Word)

Durata (cicli): 8 (80286), 10 (80386)

Descrizione: Confronta il byte o la word puntati da ES:[DI] (ES:[EDI]) con il byte o la word puntati da DS:[SI] (DS:[ESI])

Operazione: L'istruzione CMPS/SB/SW confronta il contenuto della locazione di memoria indirizzata dal registro SI (ESI) con il contenuto della locazione di memoria indirizzata dal registro DI (EDI).
L'istruzione CMPS sottrae il contenuto della locazione di memoria puntata dal registro DI (EDI) dal contenuto della locazione di memoria puntata dal registro SI (ESI). Il risultato di questa sottrazione modifica alcuni flag, senza alterare il contenuto delle due locazioni di memoria. I registri SI (ESI) e DI (EDI) vengono incrementati o decrementati in base al valore del flag Direzione (DF): se DF è a 0, SI (ESI) e DI (EDI) vengono incrementati, mentre se DF è a 1, SI (ESI) e DI (EDI) vengono decrementati. Può essere specificato nell'istruzione se il confronto riguarda un byte o una parola; SI (ESI) e DI (EDI) sono incrementati di una unità per stringhe di un byte e di due unità per stringhe di una word.
Nota: L'argomento di destra di CMPS è l'operando indicizzato dal registro DI e indirizzato utilizzando il registro ES. Questa convenzione di default non può essere modificata.

Sintassi: CMPS *stringa_sorgente,stringa_destinazione*

Flag modificati: CF, AF, PF, OF, SF, ZF

Flag indefiniti: Nessuno

Eccezioni in modalità reale: Se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, viene generata una eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, viene generata una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13, nel caso di violazione dei limiti di segmento.

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:
```
MOV     SI,OFFSET STRINGA1
MOV     DI,OFFSET STRINGA2
CMP     STRINGA1,STRINGA2
CMPS    DS:BYTE PTR[SI],ES:[DI]
LEA     ESI,STRINGA1      ;solo 80386
LEA     EDI,STRINGA2
CMPS    STRINGA1,STRINGA2
```

CWD
(Convert Word to Doubleword)

Durata (cicli): 2

Descrizione: Converte una word in una doubleword

Operazione: L'istruzione CWD converte la word con segno contenuta nel registro AX in una doubleword con segno e la memorizza nei registri DX:AX. L'operazione consiste nell'estendere il bit più significativo di AX in tutti i bit del registro DX.

Sintassi: CWD (nessun operando)

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Nessuna

Eccezioni in modalità reale: Nessuna

Nota per l'80386: Vale quanto detto per l'80286. Inoltre, esiste l'istruzione

CDQ che estende un operando di tipo doubleword, memorizzato in EAX, in un intero con segno di 64 bit memorizzato nella coppia di registri EDX:EAX.

Esempio:

```
CWD
```

DAA
(Decimal Adjust after Addition)

Durata (cicli): 3 (80286), 4 (80386)

Descrizione: Sistemazione decimale del risultato di un'addizione

Operazione: L'istruzione DAA deve essere eseguita dopo che è avvenuta la somma di due operandi BCD nel formato compattato e converte il risultato della somma, memorizzato in AL, nel formato decimale compattato, nel rispetto delle seguenti regole:

- Se i quattro bit meno significativi di AL memorizzano un valore maggiore di 9, o se il flag CF è a 1, AL viene incrementato di 6 e AF assume il valore 1; in caso contario, AF viene azzerato.
- Se il risultato della precedente operazione è maggiore di 9FH, o se il flag CF è a 1, AL viene incrementato di 60H e il flag CF viene posto a 1; in caso contrario, CF assume il valore 0.

Sintassi: DAA (nessun operando)

Flag modificati: AF, CF, SF, ZF, PF

Flag indefiniti: OF

Eccezioni in modalità protetta: Nessuna

Eccezioni in modalità reale: Nessuna

Nota per l'80386: Nessuna eccezione

Esempio:

```
MOV     AL,08
ADD     AL,03       ;il risultato in AL è 0BH
DAA                 ;il risultato in AL è 11H
```

DAS
(Decimal Adjust al after Substraction)

Durata (cicli): 3 (80286), 4 (80386)

Descrizione: Sistemazione decimale del risultato di una sottrazione

Operazione: L'istruzione DAS deve essere eseguita solo dopo che è avvenuta la sottrazione di due numeri BCD nel formato compattato e restituisce un risultato nel formato BCD compattato, nel rispetto delle seguenti regole:

- Se i quattro bit meno significativi di AL memorizzano un valore maggiore di 9, o se il flag AF è a 1, viene decrementato AL di 6 e il flag AF assume il valore 1; in caso contrario, il flag AF viene azzerato.
- Se il risultato della precedente operazione è maggiore di 9FH, o se il flag CF è a 1, viene decrementato AL di 60H e il flag CF assume il valore 1; in caso contrario, il flag CF viene azzerato.

Sintassi: DAS (nessun operando)

Flag modificati: AF, CF, SF, PF, ZF

Flag indefiniti: OF

Eccezioni in modalità protetta: Nessuna

Eccezioni in modalità reale: Nessuna

Nota per l'80386: Nessuna eccezione

Esempio:
```
MOV     AL,12
SUB     AL,03       ;il risultato in AL è OFH
DAS                 ;il risultato in AL è 09H
```

DEC
(DECrement by 1)

Durata (cicli): 2 – 7 (80286), 2 – 6 (80386)

Descrizione: Decremento di una unità

Operazione: L'istruzione DEC sottrae una unità dal contenuto (byte o word) della locazione di memoria o del registro specificati.

Sintassi: DEC destinazione

Flag modificati: SF, OF, ZF, AF, PF

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata una eccezione generale di protezione, se l'operando si trova in un segmento in cui non è permessa la scrittura oppure se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito. Inoltre, viene generata una eccezione di stack, se il registro di segmento SS contiene un indirizzo non legale.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13, se l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:

```
DEC     AX
DEC     EXTMEM
DEC     TABELLA[BX][SI]
DEC     ECX         ;solo 80386
```

DIV

(unsigned DIVision)

Durata (cicli): 14 – 25 (80286), 14 – 41 (80386)

Descrizione: Esegue la divisione senza segno di AX per un byte; esegue la divisione senza segno di DX:AX per una word

Operazione: L'istruzione DIV esegue una divisione senza segno.
Se l'operando sorgente è un byte, il registro AX viene diviso per quel byte, il quoziente viene memorizzato nel registro AL e il resto viene memorizzato nel registro AH. Se l'operando sorgente è una word, il registro DX:AX viene diviso per quella word, i 16 bit più significativi del dividendo devono essere memorizzati nel registro DX, il quoziente viene memorizzato nel registro AX e il resto viene memorizzato nel registro DX.

Sintassi: DIV *sorgente*

Flag modificati: Nessuno

Flag indefiniti: SF, AF, OF, AF, PF, CF

Eccezioni in modalità protetta: Se il divisore è 0 oppure se il quoziente è troppo grande per essere contenuto nel registro designato (AX o AL), viene generata l'interruzione INT 0. Se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, il microprocessore genera una

eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, viene generata una eccezione di stack.

Eccezioni in modalità reale: Se il divisore è 0 oppure se il quoziente è troppo grande per essere contenuto nel registro designato (AX o AL), viene generata l'interruzione INT 0. Se l'operando di tipo word è all'offset 0FFFFH, viene generata l'interruzione INT 13. La divisione in doppia precisione è caratterizzata da un dividendo di 64 bit nei registri EDX:EAX, un divisore di 32 bit in un registro o in memoria, il quoziente di 32 bit nel registro EAX e il resto di 32 bit nel registro EDX.

Esempio:
Divisione di un byte per un altro byte:
```
MOV     AL,NUM_BTE
DIV     DIVSR_BTE     ;quoziente in AL, resto in AH
```
Divisione di una word per un byte:
```
MOV     AX,NUM_WRD
DIV     DIVSR_BTE     ;quoziente in AL, resto in AH
```
Divisione di una doubleword per una word:
```
MOV     DX,NUM_MSW
MOV     AX,NUM_LSW
DIV     DIVSR_WRD     ;quoziente in AX, resto in DX
```
Solo 80386:
```
MOV     EDX,MSB_NUM
MOV     EAX,LSB_NUM
DIV     ECX           ;quoziente in EAX, resto in EDX
```

ENTER

Durata (cicli): 11 – 16 (80286), 10 – 19 (80386)

Descrizione: Prepara lo stack a contenere i parametri della procedura chiamata

Operazione: L'istruzione ENTER viene utilizzata per preparare lo stack in base alle richieste formulate, durante la chiamata di una procedura, dalla maggior parte dei linguaggi di alto livello. Il primo operando specifica il numero di byte da allocare nello stack per le variabili locali, mentre il secondo operando rappresenta il livello di annidamento della procedura in cui si entra. ENTER ricopia nello stack della procedura chiamata i puntatori relativi alle zone di stack già allocate durante le chiamate a sottoprogramma dei livelli superiori. BP (EBP) viene utilizzato come puntatore alla base dello stack

corrente. A seguito della chiamata di una procedura di livello *n* (la cui prima istruzione è ENTER *m,n*) avviene il salvataggio nello stack della procedura chiamata del BP corrente (utilizzato dalla procedura chiamante). Questo valore viene poi decrementato di due unità e la word puntata dal nuovo BP (cioè il valore di BP della procedura di livello $n-2$) viene anch'essa memorizzata nello stack della procedura chiamata. Questa operazione di sottrazione del valore di BP e di ricopiatura della word puntata dal nuovo BP continua fino a quando sono stati ricopiati tutti i valori dei BP delle $n-1$ procedure di livello superiore a quello corrente. A questo punto viene memorizzato nello stack della procedura di livello *n* anche il valore che lo stack-pointer ha al momento della chiamata della procedura di livello *n*. Da ultimo, lo stack-pointer corrente viene decrementato di *m* per riservare spazio alle variabili locali della procedura. Se il secondo operando è 0, ENTER salva BP (EBP) sullo stack, pone BP (EBP) uguale a SP (ESP) e sottrae a SP (ESP) il primo operando.

Sintassi: ENTER *word_immediata,byte_immediato*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata una eccezione di stack, se SP supera i limiti di stack in qualunque punto dell'esecuzione dell'istruzione.

Eccezioni in modalità reale: Nessuna

Nota per l'80386: Vengono utilizzati i registri di 32 bit EBP e ESP, in quanto la dimensione di una cella dello stack viene estesa da 16 a 32 bit. Infatti, se la dimensione dell'operando è 32 bit, il microprocessore utilizza il registro ESP come puntatore alla cima dello stack corrente e il registro EBP come puntatore alla zona di stack contenente i puntatori alle zone di stack già allocate durante le chiamate a sottoprogramma dei livelli superiori.

Esempio:

```
ENTER    12,0
```

HLT
(HaLT)

Durata (cicli): 2 (80286), 5 (80386)

Descrizione: Conclude l'esecuzione

Operazione: L'istruzione HLT determina la terminazione dell'esecuzione del

programma, che può ripartire solo a seguito di un interrupt esterno o di una operazione di reset. Non vengono modificati né i registri né i flag e, se l'esecuzione viene ripresa a seguito di un interrupt, i registri CS:IP puntano all'istruzione successiva a quella di HLT.

Sintassi: HLT (nessun operando)

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: HLT è un'istruzione privilegiata che genera una eccezione generale di protezione quando il livello di privilegio corrente è un valore diverso da 0.

Eccezioni in modalità reale: Nessuna

Nota per l'80386: Nessuna eccezione

Esempio:

```
HLT
```

IDIV

(Integer DIVision, signed)

Durata (cicli): 17 – 28 (80286), 19 – 43 (80386)

Descrizione: Divisione con segno di AX per un byte; divisione con segno di DX:AX per una word

Operazione: L'istruzione IDIV esegue una divisione con segno.
Se, nell'istruzione, viene specificato un operando sorgente di un byte, avviene la divisione del contenuto del registro AX per l'operando stesso, il quoziente viene memorizzato nel registro AL e il resto viene memorizzato nel registro AH. Se, invece, viene specificato un operando sorgente di una word, avviene la divisione del contenuto dei registri DX:AX per l'operando stesso. I 16 bit più significativi del dividendo devono essere memorizzati in DX, il quoziente viene memorizzato in AX e il resto viene memorizzato in DX. Il resto ha lo stesso segno del dividendo ed è sempre inferiore ad esso.

Sintassi: IDIV *sorgente*

Flag modificati: Nessuno

Flag indefiniti: SF, AF, OF, AF, PF, CF

Eccezioni in modalità protetta: Se il divisore è 0 oppure se il quoziente è troppo grande per essere contenuto nel registro designato (AX o AL), viene gene-

rata l'interruzione INT 0. Se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, il microprocessore genera una eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, viene generata una eccezione di stack.

Eccezioni in modalità reale: Se il divisore è 0 oppure se il quoziente è troppo grande per essere contenuto nel registro designato (AX o AL), viene generata l'interruzione INT 0. Se l'operando di tipo word è all'offset 0FFFFH, il microprocessore genera l'interruzione INT 13.

Nota per l'80386: L'istruzione opera anche su doubleword.
La divisione in doppia precisione presenta un dividendo di 64 bit nei registri EDX:EAX, un divisore di 32 bit in un registro o in memoria, il quoziente di 32 bit nel registro EAX e il resto di 32 bit nel registro EDX.

Esempio:
Divisione di una word con un byte:
```
MOV     AX,NUM_WRD
IDIV    DIVSR_BTE
```
Divisione di una word con una word:
```
MOV     AX,NUM_WORD
CWD                 ;converte la word in una doubleword
IDIV    DIVSR_WRD
```
Divisione di una doubleword con una word:
```
MOV     DX,NUM_MSW
MOV     AX,NUM_LSW
IDIV    DIVSR_WRD
```
Solo 80386:
```
MOV     EDX,MSB_NUM
MOV     EAX,LSB_NUM
IDIV    DIVISORE
```

IMUL
(Integer MULtiply)

Durata (cicli): 13 – 24 (80286), 9 – 41 (80386)

Descrizione: Moltiplicazione con segno

Operazione: L'istruzione IMUL esegue una moltiplicazione con segno. Se l'operando sorgente dell'istruzione è un byte, viene moltiplicato per il contenuto del registro AL, il risultato di 16 bit con segno viene memorizzato in AX e vengono azzerati i flag CF e OF, se il registro AH contiene l'estensione

in segno del registro AL; in caso contrario, i flag CF e OF assumono il valore 1. Se l'operando sorgente dell'istruzione è una word, viene moltiplicato per il contenuto del registro AX e il risultato di 32 bit con segno viene memorizzato in DX:AX (il registro DX contiene i 16 bit più significativi). I flag CF e OF vengono azzerati, se DX contiene l'estensione in segno del registro AX (si legga la nota per l'80386); in caso contrario, i due flag assumono il valore 1. Se l'istruzione IMUL contiene tre operandi, il secondo operando, che rappresenta una word di indirizzo effettivo, viene moltiplicato con il terzo operando (una word di valore immediato) e il risultato di 16 bit viene memorizzato nel primo operando (un registro con la dimensione di una word). I flag CF e OF vengono azzerati se il risultato costituisce una word con segno compresa tra −32768 e +32767; in caso contrario, i due flag assumono il valore 1.

Sintassi: IMUL *sorgente*

Flag modificati: OF, CF

Flag indefiniti: ZF, SF, AF, PF

Eccezioni in modalità protetta: Se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, viene generata una eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: L'istruzione opera anche su doubleword.
La moltiplicazione viene eseguita in doppia precisione e il risultato di 64 bit viene memorizzato nella coppia di registri EDX:EAX. In altre parole, il risultato è la moltiplicazione di un valore di 32 bit, allocato in un registro o in memoria, per il contenuto di EAX.

Esempio:

```
MOV     AL,NUMERO
IMUL    NUMERO  ;risultato in AX
MOV     AX,VALORE1
IMUL    VALORE2 ;risultato in DX:AX
MOV     EAX,0FCAB1234H   ;solo 80386
IMUL    NUMERO  ;risultato in EDX:EAX
```

IN
(INput byte or word)

Durata (cicli): 5 (80286), 5 – 6 (80386)

Descrizione: Trasferisce un byte o una word da una porta al registro accumulatore

Operazione: L'istruzione IN carica il registro AL o AX con il contenuto della porta specificata come operando e il dato così trasferito può avere una dimensione di un byte o di una word. Il programmatore può accedere ad ogni porta, identificata da un valore compreso tra 0 e 65535, collocando il numero di porta nel registro DX. Se la porta viene definita da un byte di dato, è possibile accedere solo alle porte numerate da 0 a 255.
Gli indirizzi delle porte di I/O tra 00F8H e 00FFH sono riservati alla Intel e quindi non possono essere utilizzati.

Sintassi: IN *accumulatore,porta*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata una eccezione generale di protezione quando il valore di IOPL (livello di privilegio per operazioni di Input/Output) nel registro dei flag è maggiore del livello di privilegio corrente.

Eccezioni in modalità reale: Nessuna

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:
```
IN     AL,B_P_ADR      ;trasferisce un byte in AL
IN     AX,W_P_ADR      ;trasferisce una word in AX
IN     AL,DX           ;trasferisce un byte in AL
IN     AX,DX           ;trasferisce una word in AX
IN     EAX,PORTA8      ;solo 80386
```

INC
(INCrement by 1)

Durata (cicli): 2 – 7

Descrizione: Incrementa di una unità una variabile (byte o word) o un registro con la dimensione di una word

Operazione: L'istruzione INC aggiunge una unità all'operando senza alterare il flag CF.

Sintassi: INC *destinazione*

Flag modificati: SF, OF, ZF, AF, PF

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata una eccezione generale di protezione se l'operando è memorizzato in un segmento in cui non è permessa la scrittura, oppure se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:
```
INC     AX
INC     EBX         ;solo 80386
INC     MEM_LOC
```

INS / INSB / INSW
(INput from port to String/String Byte/String Word)

Durata (cicli): 5 (80286), 5 – 8 (80386)

Descrizione: Trasferisce un byte (una word) dalla porta DX alla locazione di memoria puntata da ES:[DI]

Operazione: Con il registro DX (EDX) che contiene l'identificatore numerico di una porta di ingresso, l'istruzione INS trasferisce un elemento di stringa della dimensione di un byte o di una word nella locazione di memoria puntata da ES:[DI]. Il primo operando dell'istruzione INS specifica se il dato che viene trasferito è un byte o una word. L'operando di memoria deve essere indirizzabile dal registro ES, in quanto non è lecita, nell'istruzione, nessuna indicazione esplicita di registro di segmento.
L'istruzione INS non permette che il programmatore specifichi il numero di porta con un valore immediato; la porta può essere indirizzata solo utilizzando il registro DX.
Dopo l'esecuzione dell'istruzione, il contenuto del registro DI (EDI) avanza automaticamente; se il flag DF è a 0 (è stata eseguita l'istruzione CLD), il

registro DI (EDI) si incrementa, mentre se DF è a 1 (è stata eseguita l'istruzione STD), il registro DI (EDI) si decrementa. L'incremento o il decremento è di una unità, se è stato trasferito un byte; è di due unità se è stata trasferita una word.

Sintassi: INS *stringa__destinazione,porta*
INSB (nessun operando)
INSW (nessun operando)

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata una eccezione generale di protezione quando il valore del livello di privilegio corrente (CPL) è maggiore del valore contenuto in IOPL (livello di privilegio per operazioni di Input/Output), oppure quando l'operando destinazione si trova in un segmento su cui non è permessa la scrittura. Se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, il microprocessore genera una eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, viene generata una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: L'istruzione opera anche su doubleword. Gli indirizzi di memoria sono costituiti da indirizzi effettivi di 32 bit. Il numero di porta è contenuto nel registro DX. Gli indirizzi delle porte non vengono influenzati dalla dimensione di 32 bit delle word.

Esempio:

```
INS     STR__BTE,DX   ;trasferisce un byte
INS     STR__WRD,DX   ;trasferisce una word   La lunghezza delle stringhe
INSB                  ;trasferisce un byte           è nel registro ECS
INSW                  ;trasferisce una word
```

INT / INTO
(INTerrupt)

Durata (cicli): 23 – 167 (80286), 33 – 287 (80386)

Descrizione: Genera la chiamata ad una procedura di interruzione

Operazione: L'istruzione INT genera a livello software una chiamata ad una procedura di interrupt. L'operando immediato dell'istruzione, che è un numero compreso tra 0 e 255 inclusi, rappresenta il valore dell'indice nella ta-

bella dei descrittori di interruzione (IDT) attraverso cui vengono referenziate le procedure di interruzione. Se il microprocessore opera in modalità protetta, la IDT contiene descrittori di 8 byte, che rappresentano le porte di accesso (interrupt-gate, trap-gate, o task-gate) alle procedure, mentre se lavora in modalità reale, la IDT è un array di puntatori di 4 byte memorizzato a partire dalla locazione di memoria 00000H.
L'istruzione INTO è identica all'istruzione INT, ad eccezione del numero che identifica l'interruzione, che ha implicitamente il valore 4. Inoltre, l'interruzione stessa viene realizzata solo se il flag OF è a 1.
In modalità di indirizzamento reale, l'istruzione INT salva sullo stack, in successione, i flag, il registro CS e l'indirizzo di ritorno IP (EIP) e poi salta all'istruzione contenuta nella cella di memoria indicata dall'identificatore di interruzione.

Sintassi: INT *tipo_interruzione*
INTO (nessun operando)

Flag modificati: Se si verifica un task-switch (cioè un cambio di contesto di processo) vengono modificati tutti i flag; in caso contrario, nessuno.

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Può essere generata una eccezione generale di protezione, una eccezione di assenza del descrittore, una eccezione di stack o una eccezione di non validità del segmento di stato del processo.

Eccezioni in modalità reale: Nessuna

Nota per l'80386: La dimensione dell'operando non ha alcun effetto. Le dimensioni dei registri EIP, ESP e EFLAGS vengono dedotte dal gate.

Esempio:
```
INT     21H
INTO
```

IRET
(Interrupt RETurn)

Durata (cicli): 17 – 169 (80286), 22 – 275 (80386)

Descrizione: Ritorno da interrupt

Operazione: In modalità di indirizzamento reale, l'istruzione IRET estrae dallo stack il contenuto dei registri IP (EIP), CS e FLAGS, e ripristina l'esecuzione del codice interrotto.

In modalità protetta, l'istruzione IRET ha un comportamento che dipende dal valore del flag NT.
Se NT è a 1, il processo che esegue l'istruzione IRET cede il controllo dell'unità centrale al processo che aveva eseguito l'istruzione CALL o INT, cioè al processo che aveva generato il task-switch (il cambio di contesto di processo). IRET salva nel TSS (Task State Segment) del processo che termina l'esecuzione lo stato del processo stesso. Questo significa che alla successiva chiamata del processo, viene eseguita l'istruzione seguente a quella di IRET.
Se NT è a 0, l'istruzione IRET genera un ritorno dalla procedura di gestione dell'interrupt, senza eseguire un task-switch. Il livello di privilegio del codice sorgente, a cui il controllo della CPU ritorna, deve essere uguale o minore di quello posseduto dalla procedura di gestione dell'interrupt.

Sintassi: IRET (nessun operando)

Flag modificati: Il registro dei flag viene interamente estratto dallo stack.

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Può essere generata una eccezione generale di protezione, una eccezione di assenza del descrittore, o una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 se viene eseguita un'operazione di estrazione dallo stack quando il puntatore allo stack ha un offset di 0FFFFH.

Nota per l'80386: Viene utilizzato un program counter di 32 bit e vengono estratti dallo stack il registro EIP di 48 bit, il registro ESP di 48 bit e il registro dei flag di 32 bit.

Esempio:

```
IRET
```

J [*condizione*]

(Jump short if *condition* met)

Istruzione		Durata (cicli)	Descrizione
*	JA	7	Salto se maggiore (senza segno)
*	JAE	7	Salto se maggiore o uguale (senza segno)
*	JB	7	Salto se minore (senza segno)

	Istruzione	Durata (cicli)	Descrizione
*	JBE	7 – 9	Salto se minore o uguale (senza segno)
	JC	7 – 9	Salto se riporto
	JCXZ	8	Salto se il registro CX è a 0
*	JE	7	Salto se uguale
*	JG	7	Salto se maggiore (con segno)
*	JGE	7	Salto se maggiore o uguale (con segno)
*	JL	7	Salto se minore (con segno)
*	JLE	7	Salto se minore o uguale (con segno)
*	JNA	7	Salto se non è maggiore (senza segno)
*	JNAE	7	Salto se non è maggiore o uguale (senza segno)
*	JNB	7	Salto se non è minore (senza segno)
*	JNBE	7	Salto se non è minore o uguale (senza segno)
	JNC	7	Salto se no riporto
*	JNE	7	Salto se diverso (senza segno)
*	JNG	7	Salto se non è maggiore (con segno)
*	JNGE	7	Salto se non è maggiore o uguale (con segno)
*	JNL	7	Salto se non è minore (con segno)
*	JNO	7	Salto se non è minore o uguale (con segno)
	JNP	7	Salto se no parità
*	JNS	7	Salto se flag Segno < > 1
*	JNZ	7	Salto se flag Zero < > 1
*	JO	7	Salto se Overflow
*	JP	7	Salto se parità
*	JPE	7	Salto se parità pari
*	JPO	7	Salto se parità dispari
*	JS	7	Salto se flag Segno = 1
*	JZ	7	Salto se flag Zero = 1

La dimensione dello spiazzamento è 16/32 bit per l'80386

Operazione: L'istruzione J [condizione] trasferisce il controllo al suo operando, eseguendo salti condizionali a breve raggio, in base al tipo e al valore dei flag esaminati. L'operando, in questo caso, deve essere allocato in un intervallo compreso fra 128 byte prima e 127 byte dopo l'istruzione che segue la J [condizione]. Questa restrizione consente all'assemblatore di tradurre lo spiazzamento con un valore di un byte con segno, a partire dalla locazione successiva a quella dell'istruzione corrente.

Sintassi: J [*condizione_di_test*]

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata una eccezione generale di protezione quando lo spiazzamento referenzia una locazione esterna ai limiti del segmento di codice.

Eccezioni in modalità reale: Nessuna

Nota per l'80386: Il program counter ha una dimensione di 32 bit. Quando il controllo viene trasferito ad una istruzione del segmento di codice corrente, lo spiazzamento può essere di 8, 16 o 32 bit. Se il trasferimento invece avviene in un altro segmento di codice, è indispensabile utilizzare l'istruzione di salto incondizionato (JMP).

Esempio:

```
JA        INST_LABEL
```

JMP
(JuMP)

Durata (cicli): 7 – 183 (80286), 7 – 268 (80386)

Descrizione: Esegue un salto incondizionato

Operazione: L'istruzione JMP trasferisce il controllo del programma ad un'altra istruzione, senza memorizzare alcuna informazione di ritorno, ed esegue salti tra segmenti di codice distinti e salti diretti e indiretti all'interno del segmento di codice corrente.
I salti diretti all'interno del segmento di codice corrente utilizzano solamente l'offset di un byte, di una word o di una doubleword che viene referenziato dall'istruzione. I salti indiretti all'interno del segmento di codice corrente utilizzano, invece, il contenuto della locazione di memoria indirizzata dai byte referenziati dall'istruzione. Quando l'istruzione rappresenta un salto diret-

to all'interno del segmento, al contenuto del registro IP (EIP) viene sommato lo spiazzamento, riferito all'istruzione JMP, dell'istruzione a cui si intende trasferire il controllo del programma.
I salti tra segmenti diversi modificano il contenuto del registro CS. Per salti diretti, questa operazione viene realizzata utilizzando la seconda word referenziata dall'istruzione, mentre per salti indiretti, il registro CS viene caricato con la seconda word dell'indirizzo del dato che è indicato nell'istruzione.

Sintassi: JMP (*destinazione*)

Flag modificati: Tutti i flag vengono modificati se ha luogo un task switch (cioè un cambio di contesto di processo); in caso contrario, nessun flag viene alterato.

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Se il salto è a breve raggio (NEAR), viene generata una eccezione generale di protezione se l'offset della locazione, a cui si intende trasferire il controllo, supera i limiti del segmento di codice corrente. Se il salto a breve raggio viene realizzato da un'istruzione JMP di tipo FAR, viene generata in ogni caso una eccezione generale di protezione. Altre possibili eccezioni che possono aversi sono l'eccezione di assenza del descrittore, l'eccezione di stack e l'eccezione di non validità del segmento di stato del processo. Quando, invece, l'operando di una istruzione di salto indiretto tra segmenti diversi si trova in un registro, viene generata una eccezione di codice operativo indefinito.

Eccezioni in modalità reale: Quando l'operando di una istruzione di salto indiretto tra segmenti diversi si trova in un registro, viene generata l'eccezione di codice operativo indefinito.

Nota per l'80386: Si utilizza un program counter di 32 bit; gli operandi di tipo puntatore hanno una dimensione di 48 bit e gli spiazzamenti di 32 bit.

Esempio:
Salto diretto tra segmenti diversi:

```
JMP     FAR_LABEL
```

Salto indiretto tra segmenti diversi:

```
JMP     TABELLA[BP][DI]
JMP     TABELLA[EBP][EDI]    ;solo 80386
```

Salto diretto all'interno del segmento corrente:

```
JMP     NEAR_LABEL
```

Salto indiretto all'interno del segmento corrente:

```
JMP     WORDPTR[BX][DI]
```

LAHF
(Load AH from Flag)

Durata (cicli): 2

Descrizione: Carica nel registro AH il byte meno significativo del registro dei flag

Operazione: L'istruzione LAHF trasferisce i flag SF, ZF, AF, PF e CF nel registro AH, memorizzandoli con la seguente disposizione:

SF	ZF	x	AF	x	PF	x	CF
bit7							0

I bit in posizione 1, 3 e 5 sono indefiniti.

Sintassi: LAHF (nessun operando)

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Nessuna

Eccezioni in modalità reale: Nessuna

Nota per l'80386: Nessuna eccezione

Esempio:

```
LAHF
```

LAR
(Load Access Rights byte)

Durata (cicli): 14 – 16 (80286), 15 – 16 (80386)

Descrizione: Carica in un registro o in una cella di memoria il byte dei diritti di accesso di un descrittore

Operazione: Il secondo operando dell'istruzione LAR (un registro o una locazione di memoria della dimensione di una word) contiene un selettore. Se il descrittore che viene identificato da questo selettore è visibile dal programma, cioè se il valore maggiore tra il livello di privilegio corrente (CPL) del segmento di codice corrente e l'RPL del selettore risulta minore o uguale al livello di privilegio del descrittore (DPL), il byte dei diritti di accesso del descrittore viene caricato nel byte più significativo del primo operando, mentre il byte meno significativo viene azzerato. Se il caricamento è stato eseguito, il flag ZF assume il valore 1; in caso contrario, il flag ZF viene azzerato.

Sintassi: LAR (*byte_diritti_accesso,selettore*)

Flag modificati: ZF

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, viene generata una eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 6. In modalità di indirizzamento reale, l'istruzione LAR, infatti, non viene riconosciuta.

Nota per l'80386: Nessuna eccezione.

Esempio:
```
LAR      BDA,SELETTORE
```

LDS / LES
LSS / LFS / LGS (80386)
(Load doubleword pointer into DS/ES register)

Durata (cicli): 7

Descrizione: Carica un indirizzo di quattro byte in un registro di segmento (DS/ES) e in un registro che ha una dimensione di una word

Operazione: L'istruzione LDS/LES carica un puntatore di 4 byte, memorizzato nella locazione di memoria indicata dal secondo operando dell'istruzione, in un registro di segmento e in un registro che ha la dimensione di una word. La prima word del puntatore (offset) viene caricata nel registro che è indicato dal primo operando dell'istruzione, mentre la seconda word del puntatore viene caricata nel registro DS (con LDS) o nel registro ES (con LES).

Sintassi: LDS *destinazione,sorgente*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Possono venire generate l'eccezione generale di protezione e l'eccezione di assenza del descrittore. Se l'operando si trova al di fuori dei limiti di segmento, il microprocessore genera una eccezione generale di protezione o una eccezione di stack. Se l'operando sorgente è un registro, viene generata una eccezione di codice operativo indefinito.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 per un operando di offset 0FFFFH o 0FFFDH. Quando l'operando sorgente è un registro, il microprocessore genera una eccezione di codice operativo indefinito.

Nota per l'80386: L'istruzione opera anche su doubleword. Gli operandi di tipo puntatore sono di 48 bit. L'operazione si estende anche alle istruzioni LSS/LFS/LGS.

Esempio:

```
LDS     SI,DOUBLEWORD1
LES     BX,DOUBLEWORD2
```

LEA

(Load Effective Address offset)

Durata (cicli): 3 (80286), 2 (80386)

Descrizione: Trasferisce l'offset dell'operando sorgente nell'operando destinazione

Operazione: Il primo operando dell'istruzione LEA è il registro destinazione, che viene caricato con il valore dell'offset del secondo operando.

Sintassi: LEA *destinazione,sorgente*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata una eccezione di codice operativo indefinito se il secondo operando è un registro.

Eccezioni in modalità reale: Viene generata una eccezione di codice operativo indefinito se il secondo operando è un registro.

Nota per l'80386: L'istruzione opera anche su doubleword. Gli indirizzi di memoria sono costituiti da indirizzi effettivi di 32 bit.

Esempio:

```
LEA     AX,[BP][DI]
LEA     EBX,DATI        ;solo 80386
```

LEAVE

Durata (cicli): 5 (80286), 4 (80386)

Descrizione: Genera un ritorno da procedura per linguaggi di alto livello

Operazione: L'istruzione LEAVE esegue l'operazione inversa a quella realizzata dall'istruzione ENTER: disalloca tutte le variabili locali e pone SP uguale a BP, memorizzando nei registri il valore che possedevano al momento della chiamata della procedura.
Quando il contenuto del registro BP (EBP) viene copiato nel registro SP (ESP), lo spazio di stack utilizzato dalla procedura viene rilasciato. Nel registro BP (EBP) viene memorizzato il puntatore alla zona dati su stack del livello chiamante e l'istruzione successiva RET *nn* estrae dallo stack ogni parametro che era stato passato alla procedura chiamata.

Sintassi: LEAVE (nessun operando)

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Il microprocessore genera una eccezione di stack quando il registro BP (EBP) non punta ad una locazione interna allo stack corrente.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando un operando implicito di tipo word ha un offset di 0FFFFH.

Nota per l'80386: L'istruzione opera anche su doubleword e vengono utilizzati i registri EBP e ESP.

Esempio:
```
LEAVE
```

LGDT / LIDT
(Load Global/Interrupt Descriptor Table register)

Durata (cicli): 11 – 12

Descrizione: Carica il registro GDTR/IDTR con il contenuto di 6 byte di memoria

Operazione: L'instruzione LGDT/LIDT carica i 6 byte di memoria, che sono puntati dall'indirizzo effettivo dell'operando, nel registro della tabella dei descrittori GDTR/IDTR. Il campo LIMITE del registro della tabella dei de-

scrittori viene caricato dai primi due byte, mentre i successivi tre byte finiscono nel campo BASE del registro e l'ultimo byte viene ignorato. Queste due istruzioni possono essere eseguite solo dal sistema operativo, essendo istruzioni privilegiate.

Sintassi: LGDT *operando_memoria*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Il microprocessore genera una eccezione generale di protezione, se il livello di privilegio corrente non è 0, e una eccezione di codice operativo indefinito, se l'operando sorgente è un registro. Inoltre, se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, viene generata una eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, viene generata una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 se l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: Nessuna eccezione

Esempio:

```
LGDT    MEM_WRD
LIDT    MEM_WRD
```

LLDT

(Load Local Descriptor Table register)

Durata (cicli): 17 – 19 (80286), 20 (80386)

Descrizione: Carica il selettore nel registro LDTR

Operazione: L'operando di tipo word dell'istruzione LLDT (registro o locazione di memoria) deve contenere un selettore che punti alla GDT. Il selettore, in realtà, referenzia un descrittore che contiene, al suo interno, l'indirizzo fisico effettivo della LDT che viene caricato nel registro LDTR. LLDT è una istruzione privilegiata per cui può essere eseguita solo dal sistema operativo.

Sintassi: LLDT *operando_word*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Il microprocessore genera una eccezione generale di protezione se il livello di privilegio corrente (CPL) non è 0, oppure se il selettore che costituisce l'operando non punta alla GDT, o anche se il descrittore referenziato nella GDT non è un descrittore di LDT. Quando il descrittore di LDT che si intende referenziare non è presente nella GDT, viene generata una eccezione di assenza del descrittore. Inoltre, se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, viene generata una eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 6, in quanto l'istruzione LLDT non è riconosciuta in modalità di indirizzamento reale.

Nota per l'80386: Nessuna eccezione

Esempio:
```
LLDT    BP
```

LMSW
(Load Machine Status Word)

Durata (cicli): 3 – 6 (80286), 10 – 13 (80386)

Descrizione: Carica la word indirizzata nel registro di stato

Operazione: LMSW è un'istruzione privilegiata che può essere eseguita solo dal sistema operativo; questa istruzione carica il registro di stato con l'operando sorgente e può essere utilizzata per attivare la modalità di indirizzamento protetto. In questo caso, LMSW deve essere seguita da un'istruzione di salto alle istruzioni, interne al segmento di codice corrente, che realizzano le necessarie inizializzazioni.

Sintassi: LMSW *operando_sorgente*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Il microprocessore genera una eccezione generale di protezione, se il livello di privilegio corrente (CPL) non è 0 oppure se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, mentre se il registro di segmento contiene un indirizzo non valido, viene generata una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 se l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: Nessuna eccezione

Esempio:

```
LMSW    SP
```

LOCK (80386)
(activate the bus LOCK signal)

Durata (cicli): 0

Descrizione: Attiva il segnale BUS LOCK

Operazione: Il prefisso di istruzione LOCK permette di isolare l'area di memoria che viene specificata dall'operando destinazione dell'istruzione, attivando il segnale BUS LOCK dell'80386 e questo effetto permane per tutta la durata dell'istruzione. La locazione di memoria specificata rimane protetta fino a quando un altro processore non esegue una istruzione, che referenzia la stessa area di memoria, con il compito di disattivare l'istruzione LOCK. Le istruzioni dell'80386 che possono essere precedute dal prefisso LOCK sono le seguenti:

BT, BTS, BTR,	
BTC	memoria, registro/immediato
XCG	registro, memoria
XCG	memoria, registro
ADC, SUB,	
ADC, SBB, OR,	
XOR, AND	memoria, registro/immediato
NOT, NEG,	
INC, DEC	memoria

Sintassi: LOCK *istruzione,tipo_operando*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata una eccezione generale di protezione se l'IOPL (livello di privilegio per operazioni di Input/Output) è minore del livello di privilegio corrente.

Eccezioni in modalità reale: Nessuna

Nota per l'80386: I microprocessori 8086 e 80286 possiedono alcune istruzioni sostitutive della funzione lock che è disponibile sull'80386. Un'applicazione che è stata scritta per i precedenti microprocessori potrebbe non funzionare correttamente sull'80386.

Esempio:

```
LOCK      XCG    MEM_WORD,AX
```

LODS / LODSB / LODSW
(LOaD String/String Byte/String Word)

Durata (cicli): 5

Descrizione: Carica il byte puntato da [SI], DS:[SI] in AL; carica il byte puntato da [ESI], DS:[ESI] in AL

Operazione: Le istruzioni LODS trasferiscono un byte o una word dall'operando sorgente, puntato dal registro SI (ESI), nel registro AL o AX. Il registro SI (ESI) avanza automaticamente: se il flag Direzione è a 0 (è stata eseguita l'istruzione CLD), SI (ESI) si incrementa, mentre se il flag Direzione è a 1 (è stata eseguita l'istruzione STD), SI (ESI) si decrementa. Il passo di avanzamento è 1 per operandi di un byte e 2 per operandi di una word.

Sintassi: LODS *stringa_sorgente*
LODSB (nessun operando)
LODSW (nessun operando)

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, viene generata una eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 se l'operando di tipo word è all'offset 0FFFFH

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:

```
LEA       SI,DATO    ;se il dato è di 16 bit, il risultato è in AX
LODS
```

LOOP [*condizione*]
(LOOP control with CX counter)

Istruzione	**Descrizione**
LOOP	Ciclo se CX < > 0 (CX viene decrementato)
LOOPE	Ciclo se CX < > 0, ZF = 1 (CX viene decrementato)
LOOPNE	Ciclo se CX < > 0, ZF = 0 (CX viene decrementato)
LOOPNZ	Ciclo se CX < > 0, ZF = 0 (CX viene decrementato)
LOOPZ	Ciclo se CX < > 0, ZF = 1 (CX viene decrementato)

Nota: Si può sostituire ECX a CX

Durata (cicli): 8 (80286), 11 (80386)

Operazione: L'istruzione LOOP decrementa di una unità il registro CX (ECX) e non modifica alcun flag. Perché il ciclo venga eseguito, devono essere soddisfatte le condizioni associate all'istruzione LOOP. Il ciclo consiste in un salto a breve raggio, all'interno del segmento corrente, alla locazione di memoria referenziata dall'etichetta che è specificata nell'istruzione. La locazione di memoria a cui si intende trasferire il controllo non deve trovarsi ad una distanza maggiore di 128 byte prima o 127 byte dopo l'istruzione seguente la LOOP.

Sintassi:
LOOP *short_label*
LOOPE *short_label*
LOOPZ *short_label*
LOOPNZ *short_label*
LOOPNE *short_label*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata una eccezione generale di protezione se l'etichetta referenzia una locazione di memoria che si trova al di fuori dei limiti del segmento di codice corrente.

Eccezioni in modalità reale: Nessuna

Nota per l'80386: Nessuna eccezione

Esempio:

```
        MOV   CL,04H
AGAIN:  ADD   BX,01H
        ADD   AL,INFO[BX]
        DAA
        LOOP  AGAIN
        MOV   CX,12H
```

```
NEXT:          INC     BX
               CMP     TABELLA[BX],0
               LOOPE   NEXT
               MOV     ECX,0ABCDEH      ;solo 80386
RIPETI:        INC     BX
               MOV     AL,TABELLA_A[BX]
               ADD     AL,TABELLA_B[BX]
               MOV     TOTALE[BX],AL
               LOOPNZ  RIPETI
```

LSL
(Load Segment Limit)

Durata (cicli): 14 – 16 (80286), 20 – 26 (80386)

Descrizione: Carica in un registro il limite di segmento contenuto nel descrittore

Operazione: Se il selettore di segmento, contenuto nel secondo operando dell'istruzione LSL (registro o locazione di memoria), è visibile a livello di CPL (livello di privilegio corrente), il campo LIMITE del descrittore di segmento (una word) viene caricato nel primo operando dell'istruzione, che deve essere un registro. Se questo caricamento ha luogo correttamente, il flag Zero assume il valore 1, altrimenti il flag Zero viene azzerato.

Sintassi: LSL *limite_seg,selettore*

Flag modificati: ZF

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, viene generata una eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 6, in quanto l'istruzione LSL non è riconosciuta in modalità di indirizzamento reale.

Nota per l'80386: Nessuna eccezione

Esempio:

```
LSL            AX,SELETTORE
```

LTR
(Load Task Register)

Durata (cicli): 17 – 19 (80286), 23 – 27 (80386)

Descrizione: Carica la word indirizzata nel registro TR (task register).

Operazione: L'istruzione LTR carica il registro TR con il contenuto del registro o della locazione di memoria sorgente e marca lo stato del TSS referenziato "non libero", cioè non disponibile ad operazioni di task-switch (cambio di contesto di processo). Questa è una istruzione privilegiata per cui può essere eseguita solo dal sistema operativo.

Sintassi: LTR *operando_sorgente*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, viene generata una eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 6, in quanto l'istruzione LTR non è riconosciuta in modalità di indirizzamento reale.

Nota per l'80386: Nessuna eccezione

Esempio:

```
LTR          MEM_WRD
```

MOV
(MOVe)

Durata (cicli): 2 – 19 (80286), 2 – 22 (80386)

Descrizione: Copia l'operando sorgente nell'operando destinazione

Operazione: Esistono parecchi tipi di istruzioni MOV, ma la funzione da esse realizzata è la stessa: copiano il contenuto dell'operando sorgente nell'operando destinazione, senza distruggerlo.

Sintassi: MOV *destinazione,sorgente*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Il microprocessore può generare una eccezione generale di protezione, una eccezione di stack, oppure una eccezione di assenza di descrittore, se sta per essere caricato un registro di segmento. L'eccezione generale di protezione viene generata anche se l'operando destinazione si trova in un segmento in cui non è permessa la scrittura. Inoltre, se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, viene generata una eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: L'istruzione opera anche su doubleword. Gli indirizzi di memoria sono costituiti da un indirizzo effettivo di 32 bit o da uno spiazzamento di 32 bit.

Esempio:

Da memoria ad accumulatore:

```
MOV        AX,MEM_WRD
MOV        ECX,MYWORD;solo 80386
```

Da accumulatore a memoria:

```
MOV        MEM_BYTE,AL
```

Da registro di segmento a memoria/registro:

```
MOV        BX,ES
MOV        TABELLA[BX],SS
```

Da memoria/registro a registro di segmento:

```
MOV        ES,NEXT_WRD[SI]
MOV        DS,AX
```

Da registro a registro:

```
MOV        CX,DI
MOV        EAX,ECX;solo 80386
```

Da memoria/registro a registro:

```
MOV        AX,VAL_MEM
MOV        CX,[BP][SI]
```

Da dato immediato a registro:

```
MOV        DI,513
MOV        EAX,12345678H  ;solo 80386
```

Da dato immediato a memoria/registro:

```
MOV        TABELLA[BP][SI],25
MOV        BX,77
```

MOVS / MOVSB / MOVSW
(MOVe String/String Byte/String Word)

Durata (cicli): 5 (80286), 7 (80386)

Descrizione: Trasferisce il byte (word) contenuto in DS:[SI]in ES:[DI]; trasferisce il byte (word) contenuto in DS:[ESI] in ES:[EDI]

Operazione: Le istruzioni MOVS copiano il byte (o la word) indirizzato da SI (ESI) nell'operando destinazione di tipo byte (o di tipo word) referenziato da ES:[DI] (ES:[EDI]). L'operando destinazione deve essere indirizzabile dal registro di segmento ES, in quanto non è permessa, per l'operando destinazione, l'indicazione esplicita del registro di segmento che lo contiene (questa indicazione è invece possibile per l'operando sorgente).
Dopo che l'istruzione è stata eseguita, i registri SI (ESI) e DI (EDI) avanzano automaticamente: se il flag DF è a 0 (è stata eseguita l'istruzione CLD), i registri si incrementano, mentre se il flag DF è a 1 (è stata eseguita l'istruzione STD), i registri si decrementano. Il passo di avanzamento è 1 per operandi di un byte e 2 per operandi di una word.

Sintassi: MOVS *stringa_destinazione,stringa_sorgente*
MOVSB (nessun operando)
MOVSW (nessun operando)

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Il microprocessore genera una eccezione generale di protezione se l'operando destinazione si trova in un segmento su cui non è permessa la scrittura oppure se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, mentre se il registro di segmento contiene un indirizzo non valido, viene generata una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: Gli indirizzi effettivi di memoria sono di 32 bit. La lunghezza della stringa è contenuta nel registro ECX.

Esempio:

```
        LEA     SI,SORGENTE
        LEA     DI,ES:DESTINAZIONE
        MOV     CX,50
REP:    MOVS    DESTINAZIONE,SORGENTE
```

MOVZX / MOVSX (80386)

(MOV reg./mem to Zero/Sign eXtension 16/32-bit register)

Durata (cicli): 3 – 6

Descrizione: Trasferisce l'operando sorgente in un registro di 16 o 32 bit, eseguendo un'estensione di segno o di zero.

Operazione: Le istruzioni MOVZX e MOVSX trasferiscono il contenuto di un registro o di una locazione di memoria di 8 o 16 bit in un registro di 16 o 32 bit, memorizzando nei bit più significativi il valore 0 oppure il segno dell'operando trasferito.

Sintassi: MOVZX *registro,registro/memoria*
MOVSX *registro,registro/memoria*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata una eccezione generale di protezione se l'operando in memoria viola i limiti di segmento o i diritti di accesso, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Esempio:

```
MOVZX   EAX,BX
MOVZX   CX,AL
MOVSX   AX,BX       ;istruzione MOV semplice
MOVSX   ECX,CL
MOVSX   EAX,MEMORIA
```

MUL

(MULtiply, unsigned)

Durata (cicli): 13 – 21 (80286), 9 – 41 (80386)

Descrizione: Esegue una moltiplicazione senza segno (AX = AL × byte specificato); esegue una moltiplicazione senza segno (DX:AX = AX × word specificata)

Operazione: L'operando di tipo byte dell'istruzione MUL viene moltiplicato con il contenuto del registro AL e il risultato viene memorizzato nel registro

AX. Se AH contiene il valore 0, i flag CF e OF vengono azzerati, altrimenti assumono il valore 1.

L'operando di tipo word dell'istruzione MUL viene moltiplicato con il contenuto del registro AX e il risultato viene memorizzato nel registro DX:AX. DX contiene i sedici bit più significativi del risultato, mentre i flag CF e OF vengono azzerati se DX contiene il valore 0; in caso contario, i due flag assumono il valore 1.

Sintassi: MUL *sorgente*

Flag modificati: OF, CF

Flag indefiniti: SF, ZF, AF, PF

Eccezioni in modalità protetta: Se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, viene generata una eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, viene generata una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: L'istruzione opera anche su doubleword. La moltiplicazione in doppia precisione viene realizzata moltiplicando il contenuto del registro EAX con un registro o una locazione di memoria di 32 bit. Il risultato di 64 bit viene memorizzato nella coppia di registri EDX:EAX.

Esempio:

Moltiplicazione di un byte con una word:

```
MOV     AL,MOLTP__BTE
CBW                 ;converte il byte in AL in una word in AX
MUL     VAL__BTE
```

Moltiplicazione di un byte con un byte:

```
MOV     AL,MOLTP__BTE
MUL     VAL__BTE
```

Moltiplicazione di una word con una word:

```
MOV     AX,MOLTP__WRD
MUL     VAL__WRD;i bit più significativi in DX, quelli meno significativi in
                AX
```

Solo 80386:

```
MOV     EAX,0FCAB1234H
MUL     EBX
```

NEG
(two's complement NEGation)

Durata (cicli): 2 – 7 (80286), 2 – 6 (80386)

Descrizione: Esegue la negazione in complemento a due di un byte o una word specificata

Operazione: L'istruzione NEG complementa ciascun bit dell'operando, aggiunge 1 al risultato parziale e restituisce il risultato finale all'operando nel formato in complemento a due. Il flag Carry assume il valore 1, tranne se l'operando è 0, nel qual caso CF viene azzerato.

Sintassi: NEG *destinazione*

Flag modificati: OF, SF, ZF, AF, CF, PF

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Il microprocessore genera una eccezione generale di protezione se l'operando destinazione si trova in un segmento su cui non è permessa la scrittura oppure se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, mentre se il registro di segmento contiene un indirizzo non valido, viene generata una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:
```
NEG     AL
NEG     EBX       ;solo 80386
```

NOP
(No OPeration)

Durata (cicli): 3

Descrizione: Nessuna operazione

Operazione: NOP non esegue alcuna operazione ed è un'istruzione che occupa un byte di memoria e che modifica solo il registro IP (EIP).

Sintassi: NOP (nessun operando)

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Nessuna

Eccezioni in modalità reale: Nessuna

Nota per l'80386: Nessuna eccezione

Esempio:

```
NOP
```

NOT
(logical NOT)

Durata (cicli): 2 – 7 (80286), 2 – 6 (80386)

Descrizione: Complementa ogni bit dell'operando indirizzato (di tipo byte o word)

Operazione: L'istruzione NOT complementa ogni bit dell'operando e restituisce il risultato all'operando, senza modificare alcun flag.

Sintassi: NOT *destinazione*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Il microprocessore genera una eccezione generale di protezione se l'operando destinazione si trova in un segmento su cui non è permessa la scrittura oppure se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, mentre se il registro di segmento contiene un indirizzo non valido, viene generata una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:

```
NOT     AX
NOT     EBX     ;solo 80386
```

OR
(logical inclusive OR)

Durata (cicli): 2 – 7 (80286), 2 – 7 (80386)

Descrizione: Esegue l'operazione logica OR

Operazione: L'istruzione OR esegue l'operazione logica OR tra due operandi: le coppie di bit a 0, che occupano la stessa posizione negli operandi, forniscono come risultato 0, mentre negli altri casi il risultato è 1. Il risultato finale viene memorizzato nell'operando destinazione.

Sintassi: OR *destinazione,sorgente*

Flag modificati: CF = 0, OF = 0, SF, ZF, PF

Flag indefiniti: AF

Eccezioni in modalità protetta: Il microprocessore genera una eccezione generale di protezione se l'operando destinazione si trova in un segmento in cui non è permessa la scrittura oppure se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, mentre se il registro di segmento contiene un indirizzo non valido, viene generata una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:
```
OR      CX,DI
OR      EAX,0FFFF0000H  ;solo 80386
OR      EBX,ECX  ;solo 80386
```

OUT
(OUTput byte or word)

Durata (cicli): 3 (80286), 3 – 4 (80386)

Descrizione: Trasferisce un byte o una word alla porta referenziata dal contenuto del registro DX

Operazione: Avviene il trasferimento del contenuto del registro accumulatore (AX oppure AL) ad una delle porte di I/O numerate tra 0 e 65535 (0 e 255 nel caso di porte a 8 bit).

Sintassi: OUT *porta,accumulatore*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Se IOPL (livello di privilegio per operazioni di Input/Output, contenuto nel registro dei flag) assume un valore superiore al livello di privilegio corrente (CPL), il microprocessore genera una eccezione generale di protezione.

Eccezioni in modalità reale: Nessuna

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:

```
OUT     PORTA__WRD,AX
OUT     PORTA__BYT,AL
OUT     EDX,EAX  ;solo 80386
```

OUTS / OUTSB / OUTSW

(OUTput String/String Byte/String Word to port)

Durata (cicli): 5 (80286), 7 (80386)

Descrizione: Trasferisce un byte o una word contenuti in [SI], DS:[SI] alla porta identificata dal contenuto del registro DX.

Operazione: L'istruzione OUT trasferisce una stringa di un byte o di una word dalla locazione di memoria puntata da DS:SI alla porta puntata dall'indirizzo contenuto in DX. Il secondo operando dell'istruzione OUT definisce il tipo di dato che viene trasferito (byte oppure word).
Il registro SI viene automaticamente incrementato di una unità se il flag DF è a 1, mentre viene decrementato di una unità se il flag DF è a 0. Il passo di avanzamento è 1 per operandi di un byte e 2 per operandi di una word.

Sintassi: OUTS *porta,stringa__sorgente*
OUTSB (nessun operando)
OUTSW (nessun operando)

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Se il valore di CPL (livello di privilegio corrente) è maggiore o uguale al valore di IOPL (livello di privilegio per operazioni di Input/Output), oppure se i registri di segmento CS, DS o ES

contengono un indirizzo effettivo di memoria non lecito, viene generata una eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: L'istruzione opera anche su doubleword e anche l'indirizzo di memoria è costituito da un indirizzo effettivo di 32 bit. Il registro DX contiene i 16 bit del numero di porta e gli indirizzi delle porte di I/O non vengono influenzati dalla dimensione di 32 bit delle word. La lunghezza della stringa da trasferire è contenuta nel registro ECX.

Esempio:

```
OUTS    DX,STRGA_BTE
OUTS    DX,STRGA_WRD
OUTSB
OUTSW
```

POP

(POP word off stack to destination)

Durata (cicli): 5 (80286), 5 – 7 (80386)

Descrizione: Estrae il contenuto della locazione di memoria che si trova in cima allo stack e lo memorizza in DS, ES, SS, in una locazione di memoria, in una variabile di tipo word o in un registro

Operazione: L'istruzione POP estrae la word puntata dal registro SP e la trasferisce nell'operando destinazione, incrementando di due unità il valore del registro SP.

Sintassi: POP *destinazione*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Il microprocessore può generare una eccezione generale di protezione, una eccezione di stack o una eccezione di assenza del descrittore, se viene caricato un registro di segmento. Viene generata una eccezione di stack se il contenuto del registro SP è un valore che supera la dimensione del segmento di stack, mentre viene generata una eccezione generale di protezione se l'operando destinazione si trova in un segmento in cui non è permessa la scrittura. Se i registri di segmento CS,

DS o ES contengono un indirizzo effettivo di memoria non lecito, viene generata una eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: L'istruzione opera anche su doubleword e anche l'indirizzo di memoria è costituito da un indirizzo effettivo di 32 bit.

Esempio:
L'operando è un registro:

```
POP     CX
POP     EAX        ;solo 80386
```

L'operando è un registro di segmento:

```
POP     SS
```

POPA
(POP All general register)

Durata (cicli): 19 (80286), 24 (80386)

Descrizione: Estrae dallo stack il contenuto dei registri DI, SI, BP, SP, BX, DX, CX, AX (80286); estrae dallo stack il contenuto dei registri EDI, ESI, EBP, ESP, EBX, EDX, ECX, EAX (80386)

Operazione: L'istruzione POPA estrae tutti gli otto registri generali indicati in precedenza, ma il valore estratto per il registro SP (ESP) non viene in esso caricato.
POPA esegue l'operazione inversa rispetto a quella che era stata realizzata da PUSHA, ripristinando il contenuto dei registri che il microprocessore referenziava al momento dell'esecuzione dell'istruzione PUSHA. Il primo registro che viene estratto è DI (EDI).

Sintassi: POPA (nessun operando)

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata una eccezione di stack se l'indirizzo di inizio o di fine dello stack non è interno ai limiti di segmento.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando il valore dello stack-pointer, incrementato automaticamente dall'istruzione

POPA, diventa 0FFFFH, prima che sia stato ripristinato il contenuto di tutti i registri. Ciò avviene se precedentemente sono state effettuate singole operazioni di estrazione dallo stack.

Nota per l'80386: Esiste l'istruzione POPAD che opera su registri di 32 bit.

Esempio:
```
POPA
```

POPF
(POP Flags off stack)

Durata (cicli): 5

Descrizione: Estrae il contenuto della locazione di memoria che si trova in cima allo stack e lo trasferisce nel registro dei flag

Operazione: L'istruzione POPF utilizza i due registri SS:SP per referenziare la cima dello stack e copiare le informazioni in essa contenute nel registro dei flag. Il registro SP (ESP) viene incrementato automaticamente di due unità, mentre i flag, partendo dal bit più significativo (15) e avanzando verso il bit meno significativo (0), sono i seguenti: Indefinito, Nested task, I/O Privilege Level (2 bit), Overflow, Direzione, Interrupt, Trappola, Segno, Zero, Indefinito, Carry Ausiliario, Indefinito, Parità, Indefinito e Carry.

Sintassi: POPF (nessun operando)

Flag modificati: L'intero registro dei flag viene estratto dallo stack.

Eccezioni in modalità protetta: Viene generata una eccezione di stack quando il contenuto del registro SP supera i limiti del segmento di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando il valore dello stack-pointer, incrementato automaticamente dall'istruzione POPF, diventa 0FFFFH, prima che sia stato ripristinato il contenuto di tutti i registri. Ciò avviene se precedentemente sono state effettuate singole operazioni di estrazione dallo stack.

Nota per l'80386: Esiste l'istruzione POPFD che opera sul registro EFLAGS di 32 bit.

Esempio:
```
POPF
```

PUSH
(PUSH word onto stack)

Durata (cicli): 3 – 5 (80286), 2 – 5 (80386)

Descrizione: Memorizza nello stack uno dei registri ES, CS, SS, DS, oppure un altro registro, una locazione di memoria o un operando immediato

Operazione: L'istruzione PUSH decrementa di due unità il contenuto del registro SP (ESP) e trasferisce l'operando in cima allo stack. Nei microprocessori 80286/80386, l'istruzione PUSH SP memorizza nello stack il valore che era contenuto nel registro SP prima che l'istruzione venisse eseguita, a differenza del microprocessore 8086 che memorizza nello stack il nuovo valore del registro SP (cioè quello decrementato di due unità).

Sintassi: PUSH *sorgente*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata una eccezione di stack se il nuovo valore del registro SP viola i limiti del segmento di stack. Se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, viene generata una eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando un operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:
```
PUSH    AX
PUSH    EBX         ;solo 80386
PUSH    CS
```

PUSHA
(PUSH All general register)

Durata (cicli): 17 (80286), 18 (80386)

Descrizione: Trasferisce nello stack il contenuto dei registri AX, CX, DX, BX, SP (prima del suo aggiornamento), BP, SI, DI (80286); trasferisce nello stack il contenuto dei registri EAX, ECX, EDX, EBX, ESP (prima del suo aggiornamento), EBP, ESI, EDI (80386).

Operazione: L'istruzione PUSHA salva il contenuto degli otto registri generali – (80286) AX, CX, DX, BX, SP (prima del suo aggiornamento), BP, SI e DI oppure (80386) EAX, ECX, EDX, EBX, ESP (prima del suo aggiornamento), EBP, ESI, EDI – nello stack. Il puntatore allo stack viene decrementato di 16, per tenere conto delle otto word che sono state memorizzate sullo stack. L'ordine con cui i registri vengono salvati coincide con quello indicato, per cui l'ultimo registro salvato (DI) è anche il primo ad essere estratto.

Sintassi: PUSHA (nessun operando)

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata una eccezione di stack se l'indirizzo della prima o dell'ultima locazione di stack in cui salvare i registri supera i limiti del segmento di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando il valore dello stack-pointer, decrementato automaticamente dall'istruzione PUSHA, diventa 0000H, prima che sia stato memorizzato il contenuto di tutti i registri. Ciò avviene se non esiste sullo stack spazio sufficiente per salvare il contenuto dei registri.

Nota per l'80386: Esiste l'istruzione PUSHAD che opera su registri di 32 bit.

Esempio:

```
PUSHA
```

PUSHF
(PUSH Flags register onto the stack)

Durata (cicli): 3 (80286), 4 (80386)

Descrizione: Salva sullo stack il registro dei flag

Operazione: L'istruzione PUSHF decrementa il contenuto del registro SP (ESP) di due unità e trasferisce tutto il contenuto del registro dei flag nell'operando di tipo word indirizzato da SP (ESP), secondo una precisa disposizione. Qui di seguito vengono indicati i flag interessati al trasferimento, partendo dal bit più significativo (15) e procedendo verso il bit meno significativo (0) del registro dei flag: Indefinito, Nested Task, I/O Privilege Level (2 bit), Overflow, Direzione, Interrupt, Trappola, Segno, Zero, Indefinito, Carry Ausiliario, Indefinito, Parità, Indefinito e Carry.

Sintassi: PUSHF (nessun operando)

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata una eccezione di stack quando il nuovo valore del registro SP supera i limiti dello stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando il valore dello stack-pointer, decrementato automaticamente dall'istruzione PUSHF, diventa 0000H, prima che sia stato memorizzato il contenuto del registro dei flag. Ciò avviene se non esiste sullo stack spazio sufficiente per salvare i flag.

Nota per l'80386: Esiste l'istruzione PUSHFD che opera sul registro EFLAGS di 32 bit.

Esempio:

```
PUSHF
```

RCL / RCR / ROL / ROR
(Rotate Carry Left/Right / Overflow Left/Right)

Descrizione: Istruzioni che eseguono la rotazione dei bit dell'operando

Operazione: Ogni istruzione di rotazione sposta i bit contenuti nell'operando di memoria o nel registro specificato. L'istruzione ROL (rotazione sinistra) sposta tutti i bit a sinistra di una posizione: il bit più significativo diventa il bit meno significativo. L'istruzione ROR (rotazione destra) esegue l'operazione inversa, cioè sposta tutti i bit a destra di una posizione, per cui il bit meno significativo diventa il bit più significativo.
Le istruzioni RCL e RCR utilizzano il flag Carry nelle operazioni di rotazione. Infatti, l'istruzione RCL trasferisce il contenuto del flag Carry nel bit meno significativo dell'operando e il bit più significativo dell'operando viene trasferito nel flag Carry. L'istruzione RCR esegue la transizione inversa, cioè sposta il contenuto del flag Carry nel bit più significativo dell'operando, mentre il bit meno significativo dell'operando viene spostato nel flag Carry.
Il secondo operando delle istruzioni di rotazione indica di quante posizioni devono essere spostati i bit dell'operando sorgente. Solo se queste istruzioni utilizzano un secondo operando di valore uguale a 1, viene aggiornato il flag Overflow. Per l'istruzione RCR, il controllo di overflow viene eseguito prima della rotazione, mentre per le istruzioni RCL, ROL e ROR viene eseguito dopo. Il flag di Overflow viene aggiornato in base alle seguenti condi-

Nota: I microprocessori 80286/80386 non eseguono le rotazioni che coinvolgono più di 31 cambi di posizione di bit. In tal caso vengono considerati solo i cinque bit meno significativi del secondo operando.

Sintassi:

RCL *destinazione*,1
RCL *destinazione*,CL ;CL determina il numero di rotazioni
RCL *destinazione,contatore*
RCR *destinazione*,1
RCR *destinazione*,CL
RCR *destinazione,contatore*
ROL *destinazione*,1
ROL *destinazione*,CL
ROL *destinazione,contatore*
ROR *destinazione*,1
ROR *destinazione*,CL
ROR *destinazione,contatore*

Flag modificati: OF (solo per rotazioni singole), CF

Flag indefiniti: OF per rotazioni multiple

Eccezioni in modalità protetta: Viene generata una eccezione generale di protezione se il risultato deve essere memorizzato in un segmento in cui non è permessa la scrittura, oppure se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:

```
RCL     AH,1
RCL     MEM_BTE,VAL_ROT
RCL     DH,CL
RCR     BL,1
RCR     MEM_BTE,VAL_ROT
RCR     TABELLA[BX][DI],CL
ROL     CX,1
ROL     MEM_BTE,VAL_ROT
ROL     AX,CL
ROR     BL,1
ROR     MEM_BTE,VAL_ROT
ROR     TABELLA[BX][DI],CL
```

REP / REPZ / REPE / REPNE / REPNZ

(REPeat string operation)

Durata (cicli): 5 + 9*n* (*n*=numero di iterazioni)

Descrizione: Ripete l'operazione sulla stringa

Operazione: REP, REPE e REPNE sono prefissi che causano l'esecuzione ripetuta, un numero di volte pari al contenuto del registro CX (ECX), dell'istruzione che li segue. Nel caso delle istruzioni CMPS e SCAS, si esce dal ciclo se il prefisso è REPE e il flag ZF è a 0, oppure se il prefisso è REPNE e il flag ZF è a 1.

Sintassi: REP
REPE
REPNE

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Nessuna

Eccezioni in modalità reale: Nessuna

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:

```
LEA     DI,VALORE1     ;carica la stringa destinazione
LEA     SI,VALORE2     ;carica la stringa sorgente
REPE    CMPSB          ;ripete se uguale, in un confronto byte a byte
```

RET

(RETurn from procedure)

Durata (cicli): 15 – 55 (80286), 10 – 68 (80386)

Descrizione: Ritorno (NEAR o FAR) da procedura

Operazione: L'istruzione RET trasferisce il controllo del programma all'istruzione referenziata dall'indirizzo di ritorno che era stato salvato sullo stack da un'istruzione CALL.
Un'istruzione RET, che trasferisce il controllo del programma ad una istruzione che appartiene al segmento di codice corrente, estrae dalla cima dello stack due byte e li memorizza nel program counter. Questi due byte, infatti, costituiscono l'offset della prossima istruzione da eseguire.

Un'istruzione RET, invece, che trasferisce il controllo del programma ad una istruzione che appartiene ad un segmento di codice diverso da quello corrente, estrae dalla cima dello stack due byte e li memorizza nel program counter. Questi due byte costituiscono l'offset della prossima istruzione da eseguire. Successivamente, vengono estratti dalla cima dello stack altri due byte che vengono memorizzati nel registro CS e che costituiscono l'indirizzo del segmento di codice che contiene l'istruzione da eseguire.
L'esecuzione dell'istruzione RET, senza modifica del segmento di codice corrente, e l'aggiornamento del contenuto del puntatore allo stack, prevedono l'estrazione dei due byte in cima allo stack e il loro trasferimento nel program counter; successivamente, devono venire estratti altri due byte dallo stack e trasferiti nel puntatore allo stack. Questi ultimi due byte ripristinano il vecchio contenuto del puntatore allo stack, che era stato salvato sullo stack prima dell'istruzione CALL.
L'esecuzione dell'istruzione RET, con modifica del segmento di codice, e l'aggiornamento del contenuto del puntatore allo stack, prevede l'estrazione dei due byte in cima allo stack, che rappresentano l'offset della prossima istruzione da eseguire, e il loro trasferimento nel program counter; successivamente, vengono estratti altri due byte, che rappresentano l'indirizzo del segmento di codice contenente l'istruzione da eseguire, e vengono trasferiti nel registro CS. Infine, devono venire estratti altri due byte dallo stack e memorizzati nel puntatore allo stack. Questi ultimi due byte ripristinano il vecchio contenuto del puntatore allo stack, che era stato salvato sullo stack prima dell'istruzione CALL.

Sintassi: RET (nessun operando)

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Vengono generate in certi casi eccezioni generali di protezione, eccezioni di stack ed eccezioni di assenza di descrittore.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando implicito di tipo word è all'offset 0FFFFH.

Nota per l'80386: Il registro program counter ha una dimensione di 32 bit, mentre il registro ESP ha una dimensione di 48 bit.

Esempio:

```
RET
```

SAHF
(Store AH in Flags)

Durata (cicli): 2 (80286), 3 (80386)

Descrizione: Memorizza il contenuto del registro AH nei flag SF, ZF, xx, AF, xx, PF, xx, CF.

Operazione: Il contenuto del registro AH viene caricato nei flag indicati, che occupano, rispettivamente, le posizioni 7, 6, 4, 2 e 0.

Sintassi: SAHF (nessun operando)

Flag modificati: ZF, SF, AF, PF, CF

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Nessuna

Eccezioni in modalità reale: Nessuna

Nota per l'80386: Nessuna eccezione

Esempio:
```
SAHF
```

SAL / SAR / SHL / SHR
(Shift instructions Left/Right)

Durata (cicli): 2 – 5 (80286), 3 – 7 (80386)

Descrizione: Istruzioni che operano traslazioni di bit

Operazione: Le istruzioni SAL o SHL traslano a sinistra i bit dell'operando che è specificato nell'istruzione. In questo modo, il bit più significativo viene trasferito nel flag CF, mentre il bit meno significativo viene azzerato.
Le istruzioni SAR o SHR traslano a destra i bit dell'operando che è specificato nell'istruzione. In questo modo, il bit meno significativo viene trasferito nel flag CF. L'istruzione SAR esegue una divisione con segno e il bit più significativo non cambia, mentre l'istruzione SHR esegue una divisione senza segno e il bit più significativo viene azzerato.
L'operazione di traslazione di bit viene ripetuta un numero di volte pari al valore del secondo operando, che può essere un numero o il contenuto del registro CL.
I microprocessori 80286/80386 non eseguono traslazioni di bit maggiori di 31 e, se mai dovesse presentarsi una situazione del genere, utilizzano solo

i cinque bit meno significativi del secondo operando quale indicatore del numero di traslazioni da eseguire.
Il flag OF viene aggiornato solo nel caso di istruzioni di traslazione singola: per traslazione a sinistra, il flag OF viene azzerato se il bit più significativo del risultato è uguale al contenuto finale del flag CF, altrimenti il flag OF assume il valore 1. Nel caso di istruzione SAR, il flag OF viene azzerato per traslazione singola, mentre l'istruzione SHR pone il flag OF uguale al bit più significativo dell'operando di partenza.

Sintassi: SAL *destinazione,contatore*
SHL *destinazione,contatore*
SAL *destinazione,*1
SHL *destinazione,*1
SAL *destinazione,*CL
SHL *destinazione,*CL

Flag modificati: OF (solo per istruzioni a singola traslazione), CF, AF, OF, PF, SF

Flag indefiniti: AF

Eccezioni in modalità protetta: Viene generata una eccezione generale di protezione se il risultato deve essere memorizzato in un segmento su cui non è permessa la scrittura, oppure se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: Esistono analoghe istruzioni (con il suffisso D) che operano anche su doubleword.

Esempio:
```
SHL     MEM__BTE,1
SHL     AX,CL
SHL     TABELLA[BX][DI],CL
SAR     BL,1
SAR     MEM__BTE,VAL__SHIFT
SAR     DH,CL
SAR     CX,1
SAR     MEM__BTE,1
SAR     AX,CL
```

SBB
(SuBtract with Borrow)

Durata (cicli): 2 – 3 (80286), 2 – 7 (80386)

Descrizione: Esegue la sottazione intera con prestito

Operazione: L'istruzione SBB somma il secondo operando al flag Carry e sottrae il risultato ottenuto al primo operando. Il risultato finale viene memorizzato nel primo operando.

Sintassi: SBB *destinazione,sorgente*

Flag modificati: OF, SF, ZF, PF, CF, AF

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata una eccezione generale di protezione se il risultato deve essere memorizzato in un segmento in cui non è permessa la scrittura, oppure se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: La dimensione di una word è 32 bit.

Esempio:
```
SBB     AX,BX
SBB     DX,MEM_WRD
SBB     TABELLA[BX][DI],SI
SBB     AL,3
SBB     EAX,EBX  ;solo 80386
SBB     ECX,0ABCD1234H    ;solo 80386
```

SCAS / SCASB / SCASW
(String CompAre with String/String Byte/String Word)

Durata (cicli): 7

Descrizione: Confronta il byte contenuto in AL, o la word contenuta in AX, con il contenuto di ES:[DI] oppure di ES:[EDI]

Operazione: L'istruzione SCAS sottrae al contenuto del registro AL o AX l'operando, costituito da un byte o una word di memoria, che viene puntato

dai registri ES:DI (ES:[EDI]). Il risultato viene ignorato, ma vengono aggiornati alcuni flag.
Non è permessa nell'istruzione alcuna indicazione esplicita di registro di segmento, per cui l'operando in memoria deve essere indirizzabile dal registro ES. Dopo che il confronto ha avuto luogo, DI (EDI) viene incrementato se il flag DF è a 0; altrimenti, se il flag DF è a 1, DI (EDI) viene decrementato. Il passo di avanzamento di DI (EDI) è 1 se l'operando è di tipo byte, mentre è 2 se l'operando è di tipo word.

Sintassi: SCAS *stringa_destinazione*
SCASB (nessun operando)
SCASW (nessun operando)

Flag modificati: OF, SF, ZF, PF, CF, AF

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, viene generata una eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: Esiste l'istruzione SCASD che opera su doubleword.

Esempio:
```
MOV     CL,10H      ;confronto per 16 byte
LEA     DI,VALORE1  ;locazione della stringa
MOV     AL,'S'      ;carattere da confrontare
REPNE   SCASB       ;ripete se non è uguale
```

SGDT / SIDT

(Store Global/Interrupt Descriptor Table register)

Durata (cicli): 11,12 (80286), 9 (80386)

Descrizione: Ricopia in memoria il contenuto del registro GDTR/IDTR

Operazione: Le due istruzioni SGDT e SIDT caricano rispettivamente il contenuto del registro GDTR e del registro IDTR nei sei byte di memoria che sono puntati dall'operando destinazione. Il campo LIMITE del registro viene trasferito nella prima word dei sei byte di memoria, mentre nei successivi tre byte viene memorizzato il campo BASE del registro. L'ultimo byte è indefinito.

Sintassi: SGDT *destinazione*
SIDT *destinazione*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata una eccezione di codice operativo indefinito se l'operando destinazione è un registro, mentre viene generata una eccezione generale di protezione se il risultato deve essere memorizzato in un segmento in cui non è permessa la scrittura, oppure se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH, mentre se l'operando destinazione è un registro, viene generata una eccezione di codice operativo indefinito.

Esempio:

```
SGDT    MEM_WRD
SIDT    MEM_WRD
```

SLDT

(Store Local Descriptor Table register)

Durata (cicli): 2

Descrizione: Memorizza in una word di memoria il contenuto del registro LDTR

Operazione: L'istruzione SLDT utilizza quale operando un indirizzo effettivo che punta ad un registro o ad una locazione di memoria di 2 byte, in cui memorizza il contenuto del registro LDTR.

Sintassi: SLDT *destinazione*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata una eccezione generale di protezione se l'operando destinazione si trova in un segmento su cui non è permessa la scrittura, oppure se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, mentre se il registro di segmen-

to contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 6, in quanto, in modalità di indirizzamento reale, l'istruzione SLDT non è riconosciuta.

Esempio:
```
SLDT      BP
```

SMSW
(Store Machine Status Word)

Durata (cicli): 2

Descrizione: Memorizza la parola di stato in una word di memoria

Operazione: Nell'operando destinazione (un indirizzo effettivo, un registro di 2 byte o una locazione di memoria) viene memorizzata la parola di stato del microprocessore.

Sintassi: SMSW *destinazione*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata una eccezione generale di protezione se l'operando destinazione si trova in un segmento in cui non è permessa la scrittura, oppure se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Esempio:
```
SMSW      BP
```

STC
(SeT Carry flag)

Durata (cicli): 2

Descrizione: Flag Carry = 1

Operazione: L'istruzione STC pone a 1 il flag Carry.

Sintassi: STC (nessun operando)

Flag modificati: CF = 1

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Nessuna

Eccezioni in modalità reale: Nessuna

Nota per l'80386: Nessuna eccezione

Esempio:

```
STC
```

STD
(SeT Direction flag)

Durata (cicli): 2

Descrizione: Flag Direzione = 1

Operazione: L'istruzione STD pone a 1 il flag DF, per cui nelle successive istruzioni che operano sulle stringhe i registri SI e/o DI si autodecrementano.

Sintassi: STD (nessun operando)

Flag modificati: DF = 1

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Nessuna

Eccezioni in modalità reale: Nessuna

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:

```
STD
```

STI
(SeT Interrupt enable flag)

Durata (cicli): 2 (80286), 3 (80386)

Descrizione: Flag di abilitazione degli Interrupt = 1

Operazione: L'istruzione STI pone a 1 il flag di abilitazione degli Interrupt, per cui il microprocessore, a partire dall'istruzione successiva, è pronto a rispondere ad eventuali interrupt mascherabili esterni.

Sintassi: STI (nessun operando)

Flag modificati: IF = 1

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata una eccezione generale di protezione quando il livello di privilegio corrente (CPL) è inferiore al livello di privilegio di I/O (IOPL).

Eccezioni in modalità reale: Nessuna

Nota per l'80386: Nessuna eccezione

Esempio:

```
STI
```

STOS / STOSB / STOSW

(STOre String/String Byte/String Word)

Durata (cicli): 3 (80286), 4 (80386)

Descrizione: Memorizza il contenuto del registro AL (byte) o del registro AX (word) nella locazione di memoria puntata dalla coppia di registri ES:[DI] o ES:[EDI].

Operazione: L'istruzione STOS trasferisce il contenuto del registro AL o AX nel byte o nella word di memoria indirizzata da ES:[DI] (ES:[EDI]). L'operando destinazione deve essere indirizzabile dal registro ES, poiché non sono permesse nell'istruzione indicazioni esplicite del registro di segmento.
Il registro DI (EDI) viene incrementato automaticamente se il flag DF è a 0, mentre viene decrementato se il flag DF è a 1. Il passo di avanzamento è 1 nel caso di trasferimenti di un byte e 2 nel caso di trasferimenti di una word.

Sintassi: STS *stringa_destinazione*
STOSB (nessun operando)
STOSW (nessun operando)

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata una eccezione generale di protezione se l'operando destinazione si trova in un segmento in cui non è permessa la scrittura, oppure se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: Esiste l'istruzione STOSD che opera su doubleword.

Esempio:

```
        MOV     ECX,0FFCCAAH    ;solo 80386
        LEA     EDI,VARIABILE
        MOV     AX,'-'
REP     STOSB
```

STR
(Store Task Register)

Durata (cicli): 2 (80286), 23 – 27 (80386)

Descrizione: Memorizza il contenuto del registro TR (task register) in una word di memoria

Operazione: L'istruzione STR copia il contenuto del registro TR nella locazione di memoria o nel registro di 2 byte indirizzati dall'operando.

Sintassi: STR *destinazione*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata una eccezione generale di protezione se l'operando destinazione si trova in un segmento in cui non è permessa la scrittura, oppure se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 6, in quanto, in modalità di indirizzamento reale, l'istruzione STR non è riconosciuta.

Esempio:
```
STR BP
```

SUB
(SUBtract)

Durata (cicli): 2 – 7

Descrizione: Esegue la sottrazione intera

Operazione: L'istruzione SUB sottrae l'operando sorgente all'operando destinazione e memorizza il risultato nell'operando destinazione.

Sintassi: SUB *destinazione,operando*

Flag modificati: OF, SF, ZF, PF, AF, CF

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata una eccezione di codice operativo indefinito se l'operando destinazione è un registro, mentre viene generata una eccezione generale di protezione se il risultato deve essere memorizzato in un segmento in cui non è permessa la scrittura, oppure se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:
```
SUB     AX,BX
SUB     EAX,EDX   ;solo 80386
SUB     DX,MEM_WRD
SUB     MEM_WRD,AX
SUB     MEM_BTE,7
SUB     NUMERO,0FC981576H   ;solo 80386
```

TEST

(TEST, logical compare)

Durata (cicli): 2 – 6 (80286), 2 – 5 (80386)

Descrizione: Esegue il confronto logico

Operazione: L'istruzione TEST esegue l'operazione logica AND bit a bit su due operandi. Ogni bit del risultato ha il valore 1 se i bit corrispondenti dei due operandi sono a 1; in tutti gli altri casi, il bit del risultato viene posto a 0. Il risultato dell'operazione non viene utilizzato, ma vengono modificati alcuni flag.

Sintassi: TEST *destinazione,sorgente*

Flag modificati: OF = 0, CF = 0, SF, ZF, PF

Flag indefiniti: AF

Eccezioni in modalità protetta: Se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, viene generata una eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:

```
TEST    AX,BX
TEST    EAX,EBX  ;solo 80386
TEST    MEM__BTE,6
TEST    TABELLA[BX][DI],CX
TEST    TABELLA[BX][DI],ECX    ;solo 80386
```

VERR / VERW

(VERify a segment for Reading/Writing)

Durata (cicli): 14 – 16 (80286), 10 – 16 (80386)

Descrizione: Flag ZF = 1, se sul segmento è permessa la scrittura/lettura

Operazione: Le istruzioni VERR e VERW verificano se il segmento indirizzato dal selettore contenuto in un operando in memoria o in un registro di 2 byte può essere referenziato con il livello di privilegio corrente (CPL). L'i-

struzione stabilisce anche se sul segmento è permessa la lettura o la scrittura. Se il segmento è accessibile, il flag ZF assume il valore 1, altrimenti viene azzerato.

Sintassi: VERR *selettore_seg_leggibile*
VERW *selettore_seg_scrivibile*

Flag modificati: ZF

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, viene generata una eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 6, in quanto, in modalità di indirizzamento reale, l'istruzione VERR/VERW non è riconosciuta.

Nota per l'80386: Nessuna eccezione

Esempio:

```
VERR    BP
VERW    SELETTORE
```

WAIT

Durata (cicli): 3 (80286), 6 (minimo) (80386)

Descrizione: Mette il processore in attesa di un interrupt esterno

Operazione: L'istruzione WAIT impone lo stato di attesa al microprocessore fino all'arrivo di un interrupt esterno.

Sintassi: WAIT (nessun operando)

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Viene generata l'interruzione INT 7, se il flag TS (Task Switch) nella parola di stato del microprocessore è a 1, oppure l'interruzione INT 16 se l'80287 ha scoperto un errore numerico non mascherabile.

Eccezioni in modalità reale: Viene generata l'interruzione INT 7, se il flag TS (Task Switch) nella parola di stato del microprocessore è a 1, oppure l'in-

terruzione INT 16 se l'80287 ha scoperto un errore numerico non mascherabile.

Nota per l'80386: Stesse eccezioni dell'80286

Esempio:

```
WAIT
```

XCHG
(eXCHanGe)

Durata (cicli): 3 – 5

Descrizione: Scambia byte o word

Operazione: L'istruzione XCHG scambia il byte o la word dell'operando sorgente con l'operando destinazione dello stesso tipo.

Sintassi: XCHG *destinazione,sorgente*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, viene generata una eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:

```
XCHG    AX,BX
XCHG    DH,DATO_WRD
XCHG    AL,DL
XCHG    EAX,EBX  ;solo 80386
```

XLAT
(transLATe)

Durata (cicli): 5

Descrizione: Esegue una conversione di byte mediante tabella (table lookup)

Operazione: L'istruzione XLAT effettua una conversione di byte mediante tabella di conversione. Il registro AL contiene un indice senza segno della tabella che è indirizzata da DS:BX (DS:[EBX]). Il byte indirizzato da DS:BX (DS:[EBX]) viene copiato in AL.

Sintassi: XLAT *tabella_di_conversione*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito, viene generata una eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:

```
LEA     BX,TABELLA   ;locazione della tabella
MOV     AL,INDICE    ;offset nella tabella
XLAT    TABELLA      ;valore restituito in AL
```

XOR

(logical eXclusive OR)

Durata (cicli): 2 – 3 (80286), 2 – 7 (80386)

Descrizione: Esegue l'OR esclusivo logico

Operazione: L'istruzione XOR confronta ogni bit degli operandi sorgente e destinazione e su di essi esegue l'operazione di OR eslusivo (XOR). Il risultato contiene bit a 0 in quelle posizioni dove i bit confrontati sono entrambi a 1 o a 0, mentre contiene bit a 1 in quelle posizioni dove i bit confrontati dei due operandi hanno valori opposti.

Sintassi: XOR destinazione,sorgente

Flag modificati: OF = 0, CF = 0, SF, ZF, PF

Flag indefiniti: AF

Eccezioni in modalità protetta: Se i registri di segmento CS, DS o ES contengono un indirizzo effettivo di memoria non lecito oppure se il risultato

viene memorizzato in un segmento su cui non è permessa la scrittura, viene generata una eccezione generale di protezione, mentre se il registro di segmento contiene un indirizzo non valido, il microprocessore genera una eccezione di stack.

Eccezioni in modalità reale: Viene generata l'interruzione INT 13 quando l'operando di tipo word è all'offset 0FFFFH.

Nota per l'80386: L'istruzione opera anche su doubleword.

Esempio:
```
XOR     AX,MEM_WRD
XOR     AH,BL
XOR     TABELLA[BX][DI],AX
XOR     EAX,ECX             ;solo 80386
XOR     EDX,0FCAD1579H ;solo 80386
```

3.4 Set di istruzioni dell'80386

BSF/BSR

(Bit Scan Forward/Reverse)

Durata (cicli): 10 + 3*n*

Descrizione: Scansione di bit in avanti o all'indietro

Operazione: Le istruzioni BSF e BSR esaminano l'operando sorgente (16 o 32 bit) e memorizzano nel registro destinazione il valore numerico della posizione del primo bit a 1. BSF (scansione dei bit in avanti) esamina i bit dell'operando sorgente da destra a sinistra, partendo dal bit meno significativo (posizione 0). BSR (scansione dei bit all'indietro) esamina i bit dell'operando sorgente da sinistra a destra, partendo dal bit più significativo, cioè il sedicesimo (posizione 15) nel caso di un operando sorgente di 16 bit, o il trentaduesimo (posizione 31) nel caso di un operando sorgente di 32 bit. Inoltre, il flag ZF assume il valore 1 se l'intera word contiene solo bit a 0, mentre se esiste almeno un bit a 1, il flag ZF viene azzerato. Se non esiste alcun bit a 1, il valore del registro destinazione risulta indefinito.

Sintassi: BSF *reg_destinazione,operando_sorgente*
BSR *reg_destinazione,operando_sorgente*

Flag modificati: ZF

Flag indefiniti: OF, SF, AF, PF, CF

Eccezioni in modalità protetta: Viene generata una eccezione generale 13 se l'operando non può essere utilizzato a causa di una violazione dei limiti di segmento o dei diritti di accesso.

Eccezioni in modalità reale: Viene generata una eccezione generale 13 quando il riferimento all'operando supera i limiti di segmento (0FFFFH).

Esempio:
```
BSF     AX,MEM_WORD
BSR     EAX,ECX
```

BT / BTS / BTR / BTC
(Bit Test/Set/Reset/Complement)

Durata (cicli): 3, 13

Descrizione: Le operazioni realizzate da queste istruzioni interessano i singoli bit di un registro o di una locazione di memoria.

Operazione: Le istruzioni BT, BTS, BTR operano su uno dei 16 o 32 bit di cui si compone la stringa che è memorizzata in un registro o in una locazione di memoria.
L'istruzione specifica la posizione del bit su cui l'operazione deve essere eseguita, mediante l'indicazione del valore immediato oppure referenziando un registro generale che contiene il valore numerico dello spiazzamento del bit selezionato, che può variare da 0 a 31 per stringhe di 32 bit e da 0 a 15 per stringhe di 16 bit.
Innanzitutto, le istruzioni BT, BTS, BTR e BTC memorizzano nel flag Carry il valore del bit selezionato e assegnano un nuovo valore a quest'ultimo in accordo con il tipo di operazione da esse realizzata, ad eccezione dell'istruzione BT che esegue solamente un test sul valore del bit.

Sintassi: BT *reg/mem,selettore*
BTS *reg/mem,selettore*
BTR *reg/mem,selettore*
BTC *reg/mem,selettore*

Flag modificati: CF assume il valore del bit selezionato

Flag indefiniti: OF, SF, ZF, AF, PF

Eccezioni in modalità protetta: Viene generata una eccezione generale 13 se l'operando non può essere utilizzato a causa di una violazione dei limiti di segmento o dei diritti di accesso.

Eccezioni in modalità reale: Viene generata una eccezione generale 13 quando il riferimento all'operando supera i limiti di segmento (0FFFFH).

Esempio:

```
BT      AL,BL           ;BL  = offset
BT      MEM_WRD,0BH     ;0BH = offset immediato
BTS     AX,CL           ;CL  = offset
BTS     INFO[BX],7H     ;7H  = offset immediato
BTR     AL,AL           ;AL  = offset
BTR     TAB[SI],0EH     ;0EH = offset immediato
BTC     BL,AL           ;AL  = offset
BTC     BUFF[BX],2H     ;2H  = offset immediato
```

IBTS
(Insert BiT String)

Durata (cicli): 12/19

Descrizione: Inserisce alcuni bit in un registro o in una locazione di memoria

Operazione: L'istruzione IBTS riporta i bit meno significativi di un registro in un altro registro o in una locazione di memoria, senza alterare i bit che sono memorizzati nella parte non interessata all'inserimento.
L'istruzione IBTS dispone di quattro operandi: l'indirizzo di base della stringa di bit (contenuto in un registro o in una locazione di memoria), l'offset del primo bit della sottostringa da inserire (contenuto nel registro EAX oppure nel registro AX per operandi di 16 bit), la lunghezza della sottostringa da inserire (contenuta nel registro CL) e il registro generale in cui viene effettuato l'inserimento.

Sintassi: IBTS *base,offset,lunghezza,sorgente*
IBTS *reg/mem,*(E)AX,CL,*reg*

Flag modificati: OF, SF, ZF, AF, PF

Flag indefiniti: CF

Eccezioni in modalità protetta: Il microprocessore genera una eccezione generale 13 se l'operando non può essere utilizzato a causa di una violazione dei limiti di segmento o dei diritti di accesso.

Eccezioni in modalità reale: Viene generata una eccezione generale 13 quando il riferimento all'operando supera i limiti di segmento (0FFFFH).

Esempio:

```
IBTS    BX,AX,CL,DX
```

MOV CR*n*
(MOVe Control Registers)

Durata (cicli): 2 – 4

Descrizione: Carica il registro CR*n*; memorizza il contenuto del registro CR*n* in un altro registro

Operazione: Queste istruzioni caricano i registri di controllo (CR*n*), oppure memorizzano il valore in essi contenuto in un altro registro, utilizzando sempre un operando di 32 bit. Per l'80386 sono definiti solo tre registri di controllo: CR0, CR2 e CR3.

Sintassi: MOV CR*n,operando_sorgente*
MOV *destinazione,*CR*n*
(*n* = 0, 2, 3)

Flag modificati: OF, ZF, SF, PF, AF

Flag indefiniti: CF

Eccezioni in modalità protetta: Sono istruzioni privilegiate, per cui se vengono eseguite ad un livello di privilegio diverso da 0, si genera una violazione di protezione.

Eccezioni in modalità reale: Nessuna

Esempio:
```
MOV     CR2,EAX
MOV     EBX,CR3
```

MOV DR*n*
(MOVe Debug Registers)

Durata (cicli): 2 – 4

Descrizione: Carica il registro di debug; memorizza il contenuto del registro di debug in un altro registro

Operazione: Queste istruzioni caricano nel registro di debug un valore di 32 bit oppure memorizzano il contenuto (32 bit) del registro di debug in un altro registro.

Sintassi: MOV DR*n,registro*
MOV *registro,*DR*n*

Flag modificati: OF, ZF, SF, AF, PF

Flag indefiniti: CF

Eccezioni in modalità protetta: Sono istruzioni privilegiate, per cui se vengono eseguite ad un livello di privilegio diverso da 0, si genera una violazione di protezione.

Eccezioni in modalità reale: Nessuna

Esempio:

```
MOV     DR4,EAX
MOV     EBX,DR5
```

MOV TR*n*
(MOVe Test Registers)

Durata (cicli): Carica il registro di ricerca; memorizza il contenuto del registro di ricerca in un altro registro

Operazione: Queste istruzioni caricano i registri di ricerca oppure memorizzano il contenuto dei registri di ricerca in un altro registro. Per l'80386, sono definiti solo due registri di ricerca: TR6 e TR7.

Sintassi: MOV TR*n*,*registro*
MOV *registro*,TR*n*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Sono istruzioni privilegiate, per cui se vengono eseguite ad un livello di privilegio diverso da 0, si genera una violazione di protezione.

Eccezioni in modalità reale: Nessuna

Esempio:

```
MOV     TR6,EAX
```

SET [*condizione*]
(byte SET on condition)

Istruzione	Durata (cicli)	Descrizione
SET*cond.*	4 – 5	Azzera o pone al valore unitario l'operando di un byte, in base all'esito del test condizionale
SETO	4 – 5	Overflow
SETNO	4 – 5	No overflow
SETB	4 – 5	Minore (senza segno)
SETNAE	4 – 5	No maggiore o uguale (senza segno)
SETNB	4 – 5	No minore (senza segno)
SETAE	4 – 5	Maggiore o uguale (senza segno)
SETE	4 – 5	Uguale
SETZ	4 – 5	Zero
SETNE	4 – 5	No uguale
SETNZ	4 – 5	No zero
SETBE	4 – 5	Minore o uguale (senza segno)
SETNA	4 – 5	No maggiore (senza segno)
SETNBE	4 – 5	No minore o uguale (senza segno)
SETA	4 – 5	Maggiore (senza segno)
SETS	4 – 5	Segno
SETNS	4 – 5	No segno
SETP	4 – 5	Parità
SETPE	4 – 5	Parità pari
SETNP	4 – 5	No parità
SETPO	4 – 5	Parità dispari
SETL	4 – 5	Minore (con segno)
SETNGE	4 – 5	No maggiore o uguale (con segno)
SETNL	4 – 5	No minore (con segno)
SETGE	4 – 5	Maggiore o uguale (con segno)
SETLE	4 – 5	Minore o uguale (con segno)
SETNG	4 – 5	No maggiore (con segno)
SETNLE	4 – 5	No minore o uguale (con segno)
SETG	4 – 5	Maggiore (con segno)

Operazione: Le istruzioni SET assegnano il valore 0 oppure 1 al loro unico operando destinazione, di un byte (un registro o una locazione di memoria), in base ad una delle 16 condizioni definite per l'80286: se SETcond è vera, allora reg/mem = 1, altrimenti reg/mem = 0.

Sintassi: SET*cond reg/mem*

Flag modificati: Nessuno

Flag indefiniti: Nessuno

Eccezioni in modalità protetta: Sono istruzioni privilegiate, per cui se vengono eseguite ad un livello di privilegio diverso da 0, si genera una violazione di protezione.

Eccezioni in modalità reale: Nessuna

Esempio:
SETNO

SHLD / SHRD
(SHift Left/Right Double)

Durata (cicli): 3 – 7

Descrizione: Trasla il valore in doppia precisione a sinistra/destra

Operazione: Le istruzioni SHLD/SHRD traslano a sinistra o a destra un valore espresso in doppia precisione per produrre un valore in singola precisione. Il contenuto del registro o della locazione di memoria (primo operando) viene traslato di un certo numero di posizioni in base al valore di un contatore (secondo operando). Questi spostamenti differiscono dagli spostamenti in singola precisione, in quanto gli spostamenti in doppia precisione utilizzano un operando, che è un registro, per fornire i bit da inserire. Gli spostamenti in singola precisione, invece, inseriscono solo zeri o solo uni. L'operando registro, o la locazione di memoria, specifica l'operando di ingresso, il campo registro specifica i bit da inserire, mentre il registro CL (o una costante) indica il numero di spostamenti da compiere.
Nel caso di istruzione SHLD, l'operando reg/mem (un registro o una locazione di memoria) contiene i bit più significativi di un valore in doppia precisione e l'operando reg (un registro) contiene i bit meno significativi. L'operando reg/mem viene spostato a sinistra e i bit più significativi dall'operando reg vengono inseriti a destra (posizioni meno significative). Il risultato viene memorizzato nell'operando reg/mem.
Nel caso di istruzione SHRD, l'operando reg/mem contiene i bit meno significativi di un valore in doppia precisione e l'operando reg contiene i bit più significativi. L'operando reg/mem viene spostato a destra e i bit meno significativi dall'operando registro vengono inseriti a sinistra (posizioni più significative). Il risultato viene memorizzato nell'operando reg/mem.

Sintassi: SHLD *reg/mem,reg,immediato*
SHRD *reg/mem,reg,*CL

Flag modificati: CF assume il valore dell'ultimo bit estratto; OF assume il valore 1 se lo spostamento dell'ultimo bit ha causato overflow; SF, ZF e PF vengono aggiornati in base al valore del risultato

Flag indefiniti: AF

Eccezioni in modalità protetta: Nessuna

Eccezioni in modalità reale: Nessuna

Esempio:

```
SHLD    AX,BL,0AH ;0AH = contatore immediato di shift
SHRD    AL,BL,CL  ;CL = contatore di shift
```

XBTS

(eXtract BiT String)

Durata (cicli): 6 – 13

Descrizione: Estrae una stringa di bit e la memorizza in un registro, allineandola a destra

Operazione: L'istruzione XBTS preleva una sottostringa di bit e la memorizza in un registro specificato, eseguendo l'allineamento a destra e azzerando i bit più significativi.
L'istruzione XBTS ha quattro operandi: l'indirizzo di base della stringa di bit (indirizzo di un registro o di una locazione di memoria), l'offset del primo bit della sottostringa da estrarre (contenuto nel registro EAX o nel registro AX per operandi di 16 bit), la lunghezza della sottostringa (contenuta nel registro CL) e il registro generale in cui deve essere salvato il valore estratto.

Sintassi: XBTS *dest,base,offset,lunghezza*
XBTS *reg,reg/mem,*(E)AX,CL

Flag modificati: OF, SF, ZF, AF, PF

Flag indefiniti: CF

Eccezioni in modalità protetta: Viene generata una eccezione generale 13 se l'operando non può essere utilizzato a causa di una violazione dei limiti di segmento o dei diritti di accesso.

Eccezioni in modalità reale: Viene generata una eccezione generale 13 quando il riferimento all'operando supera i limiti di segmento (0FFFFH).

Esempio:

```
XBTS    AX,BX,AX,CL
```

4

I coprocessori matematici 80287/80387

Il Capitolo 4 tratta alcune caratteristiche funzionali dei coprocessori matematici 80287/80387 che risultano utili a chi programma in linguaggio assemblatore.
In particolare, viene esaminata la struttura e la funzione dello stack per operazioni in virgola mobile, costituito da otto registri di 80 bit ciascuno che possono contenere un numero nel formato in virgola mobile, dei tre registri di 16 bit che definiscono l'ambiente 80287/80387 (*status word, control word* e *tag word*) e dei quattro registri di 16 bit contenenti i puntatori all'istruzione e al dato correnti (*instruction pointer* e *data pointer*). Inoltre, in questo capitolo viene affrontata una breve discussione sulle operazioni che il chip 80287/80387 realizza, sulla precisione e sulla velocità delle sue elaborazioni numeriche, sulla sua capacità di gestire eventuali eccezioni e, da ultimo, sul suo insieme di istruzioni.

4.1 Funzioni dell'80287/80387

Il coprocessore matematico 80287/80387 opera in parallelo con il microprocessore 80286/80386 e molte sue istruzioni codificano a livello di microprogramma alcune operazioni in virgola mobile, che risultano così estremamente rapide ed efficaci.
Quando l'80286/80386 deve eseguire un'istruzione che opera su numeri espressi nella notazione in virgola mobile (floating point), invia all'80287/80387 il relativo codice operativo e gli indirizzi di memoria degli operandi. In questo modo, l'80286/80386 evita di eseguire l'istruzione, ma

si affida, per la risoluzione dei calcoli numerici, all'80287/80387 che si serve di un canale dati dedicato, interno all'80286/80386, per richiedere l'accesso alla memoria. Queste richieste vengono accolte se sono soddisfatti i requisiti sulla protezione, mentre, in caso contrario, vengono generate eccezioni. In alcune operazioni, l'80286/80386 deve rimanere in stato di attesa fino a quando il coprocessore matematico restituisce il risultato delle proprie elaborazioni, e questa condizione può essere facilmente ottenuta utilizzando le istruzioni 80286/80386 WAIT o FWAIT.

STACK PER OPERAZIONI IN VIRGOLA MOBILE

Lo stack dell'80287/80387 si compone di otto registri di 80 bit ciascuno, suddivisi in tre campi (Figura 4.1) e questa configurazione ben si adatta al tipo di formato che viene utilizzato per i numeri reali in tutti i calcoli eseguiti dal coprocessore.
I singoli elementi dello stack possono essere indirizzati implicitamente o esplicitamente; alcune istruzioni in virgola mobile referenziano, infatti, particolari registri dello stack (a meno di esplicite controindicazioni) come ad esempio:

FSQRT Questa istruzione estrae il contenuto del registro ST(0), che si trova in cima allo stack dell'80287/80387, ne esegue la radice quadrata e rimette il risultato così ottenuto nello stack.

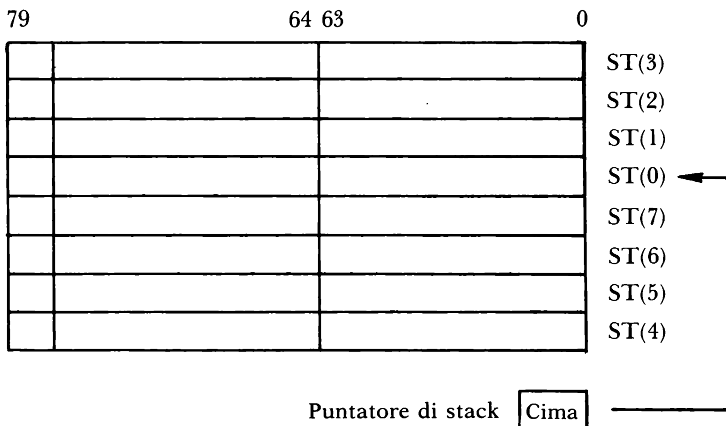


Figura 4.1 Stack dell'80287/80387

Altre istruzioni permettono al programmatore di selezionare un elemento dello stack, come nel seguente caso:

FST(7) Questa istruzione estrae il contenuto del registro ST(0), che si trova in cima allo stack dell'80287/80387, e lo memorizza nel settimo registro dello stack.

PAROLA DI STATO

La parola di stato (status word) dell'80287/80387 (Figura 4.2) riflette la condizione di funzionamento del coprocessore e si compone di due campi: un campo è riservato ai flag che riguardano la presenza di eccezioni, mentre l'altro campo contiene i flag che indicano lo stato del coprocessore. Esiste un'istruzione dell'80287/80387 che permette di memorizzare in una locazione di memoria la parola di stato, in modo che il programmatore la possa esaminare, utilizzando le relative istruzioni dell'80286/80386.

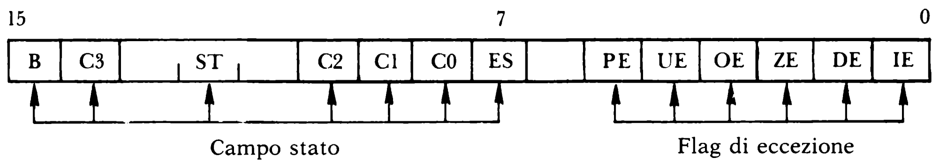
PAROLA DI CONTROLLO

La parola di controllo (control word) dell'80287/80387 (Figura 4.3) contiene gli indicatori di eccezione, di abilitazione delle interruzioni e alcuni bit di controllo.

Indicatori di eccezione

Gli indicatori di eccezione, che sono presenti nella parola di controllo, indicano quali eccezioni devono essere gestite con le usuali azioni correttive dell'80287/80387 (si parla di eccezioni mascherate) e quali eccezioni devono portare l'80287/80387 a generare segnali di interrupt (si parla di eccezioni non mascherate). Gli indicatori di eccezione sono:

[PM]	Indicatore di precisione
[UM]	Indicatore di Underflow
[OM]	Indicatore di Overflow
[ZM]	Indicatore di divisione per zero
[DM]	Indicatore di operando non normalizzato
[IM]	Indicatore di operazione non valida



[B] Questo bit indica se l'80287/80387 sta eseguendo un'istruzione oppure se è inattivo

[ST] Questi tre bit indicano quale degli otto elementi dello stack si trova in cima.
Valori di ST:
000 – L'elemento 0 è in cima allo stack
001 – L'elemento 1 è in cima allo stack
.
.
.
111 – L'elemento 7 è in cima allo stack

[C3,C2,C1,C0] Questo campo di quattro bit contiene alcune informazioni addizionali sull'elemento che si trova in cima allo stack, e in base ad esse vengono eseguiti salti condizionati (Si consultino le istruzioni dell'80287/80387 FCOM, FCOMP, FCOMPP, FTST, FXAM e FPREM, per avere maggiori dettagli sul significato dei bit C3, C2, C1 e C0).

[IR] Questo bit (richiesta di interrupt) indica che l'80287/80387 intende interrompere l'80286/80386.

[PE] Questo flag (Eccezione di Precisione) assume il valore 1 se il risultato deve essere arrotondato perché possa essere espresso nel formato in virgola mobile.

[UE] Questo flag (Eccezione di Underflow) assume il valore 1 quando il risultato di una elaborazione è troppo piccolo per essere memorizzato in forma normalizzata nell'operando destinazione.

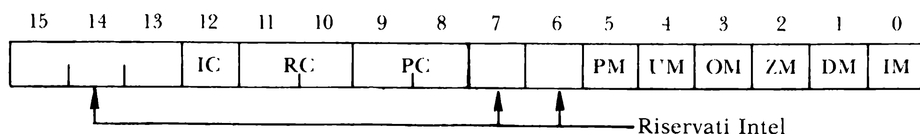
[OE] Questo flag (Eccezione di Overflow) assume il valore 1 quando il risultato di una elaborazione è troppo grande per essere memorizzato con il formato in virgola mobile nell'operando destinazione.

[ZE] Questo flag (Eccezione di Divisione per zero) indica se si è cercato di dividere un numero (diverso da zero) per zero.

[DE] Questo flag (Eccezione di Operando non normalizzato) indica se una istruzione ha cercato di operare su un operando espresso in forma non normalizzata.

[IE] Questo flag (Eccezione di Operazione non valida) indica se si è cercato di eseguire una operazione illecita, come ad esempio la radice quadrata di un numero negativo.

Figura 4.2 Parola di stato dell'80287/80387



[IC] Questo flag (Controllo di infinito) specifica come l'80287/80387 tratta i valori numerici infiniti. Se IC è a 0, il coprocessore non distingue l'infinito negativo da quello positivo, mentre se IC è a 1, questa distinzione viene correntemente adottata.

[RC] Questo flag (Controllo di arrotondamento) indica in quale delle quattro direzioni vengono arrotondati i valori numerici dei risultati: più vicino, per eccesso, per difetto, verso il valore zero.

[PC] Questo flag (Controllo di Precisione) indica quale dei tre tipi di precisione è stata scelta: temporary real (reale temporaneo, risultato di 64 bit), long real (reale lungo, risultato di 53 bit) oppure short real (reale corto, risultato di 24 bit).

Figura 4.3 Parola di controllo dell'80287/80387

PAROLA DEGLI INDICATORI

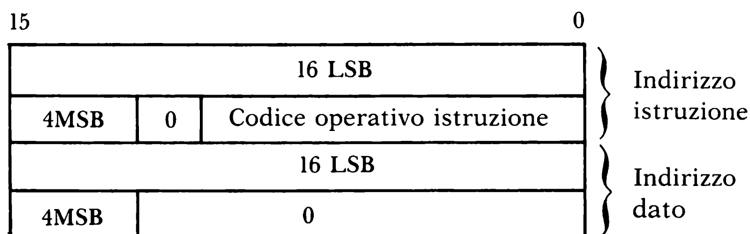
La parola degli indicatori (tag word) descrive il contenuto degli elementi dello stack e si compone dei seguenti indicatori:

TAG(7) TAG(6) TAG(5) TAG(4) TAG(3) TAG(2) TAG(1) TAG(0)

Valori degli indicatori:
00 = Valido (normale o non normale)
01 = Zero (vero)
10 = Speciale (non un numero, infinito o non normalizzato)
11 = Vuoto

PUNTATORI DI ECCEZIONE

I puntatori di eccezione permettono al programmatore di scrivere propri gestori di eccezione. Quando l'80287/80387 esegue un'istruzione, l'indirizzo dell'istruzione viene memorizzato nei puntatori di eccezione e se l'istruzione stessa referenzia un operando in memoria, viene memorizzato anche l'indirizzo di questo operando. Il programmatore può scrivere un gestore di eccezione per memorizzare questi puntatori in memoria e ottenere alcune



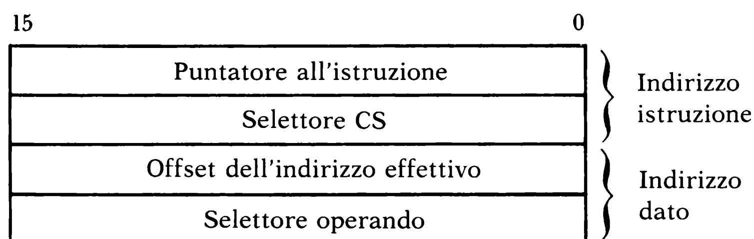


Figura 4.4 Formati dei puntatori all'istruzione e ai dati dell'80287/80387. Il chip 80387 supporta un'interfaccia dati di 32 bit con il bus di sistema

informazioni che riguardano l'istruzione che ha causato l'errore. Questa informazione viene memorizzata in due formati (Figura 4.4), a seconda che sia attiva la modalità di indirizzamento reale oppure sia attiva la modalità di indirizzamento protetta.

TIPI DI DATI

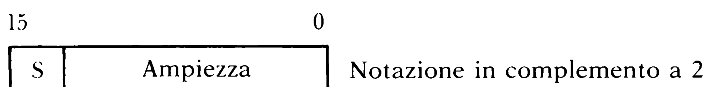
L'80287/80387 supporta sette differenti tipi di dati, ai cui valori numerici è possibile accedere utilizzando tutti i modi di indirizzamento standard che sono definiti sull'80286/80386.
In tutti i sette formati di dati, il segno del numero viene sempre memorizzato nel bit più significativo (cioè nel primo bit a partire da sinistra) del campo. Nel libro viene adottata la seguente notazione:

S	bit di segno (0 = positivo, 1 = negativo)
d17 – d0	cifre decimali memorizzate due per byte

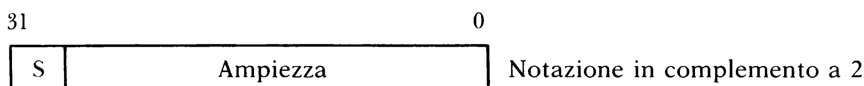
UD indefinito, senza significato
^ posizione della virgola binaria implicita
[ ] bit intero del campo mantissa: memorizzato esplicitamente nel formato temporary real e implicito nei formati short e long real.

Numeri interi binari
I tre formati di memorizzazione di un dato intero binario sono identici, ad eccezione del numero di bit che definisce il campo di valori che possono essere assunti dal dato. Il bit più significativo contiene sempre il segno del numero e i numeri negativi vengono rappresentati nella notazione in complemento a due. Il numero zero possiede il segno positivo. Un *word integer*, nell'80287/80387, possiede lo stesso formato con cui nell'80286/80386 vengono rappresentati i dati interi con segno di 16 bit.

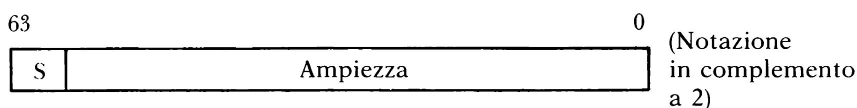
Word integer (campo di valori: $-32768 \leq X \leq +32767$)



Short integer (campo di valori: $-2 \times 10^9 \leq X \leq +2 \times 10^9$)

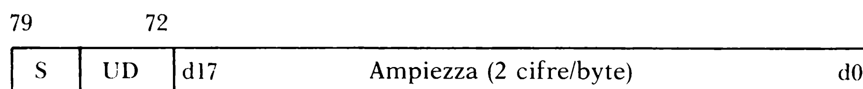


Long integer (campo di valori: $-9 \times 10^{18} \leq X \leq +9 \times 10^{18}$)



Notazione decimale compattata
La notazione decimale compattata viene utilizzata per memorizzare numeri interi decimali, con due cifre memorizzate, o compattate, in ogni byte. Il bit di segno stabilisce se il numero è negativo o positivo e le cifre devono assumere un valore compreso tra 0H e 9H, inclusi.

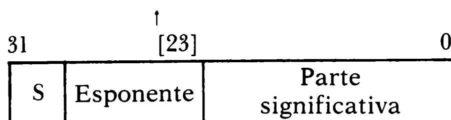
Decimale compattato (campo di valori: $-99..99 \leq X \leq +99..99$ (18 cifre))



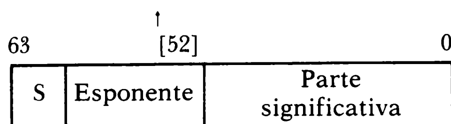
Formati Short real, Long real e Temporary real

I formati di dato short real e long real esistono solo in memoria. Quando un numero memorizzato in uno di questi formati viene caricato nello stack, viene convertito a livello hardware nel formato temporary real.

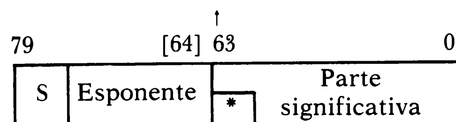
Short real (campo di valori: 0, $1.2 \times 10^{-38} \leq X \leq 3.4 \times 10^{38}$)



Long real (campo di valori: 0, $2.3 \times 10^{-308} \leq X \leq 1.7 \times 10^{308}$)



Temporary real (campo di valori: 0, $3.4 \times 10^{-4932} \leq X \leq 1.1 \times 10^{4932}$)



Valori dell'esponente (normalizzati):

Short real: 127 (7FH)
Long real: 1023 (3FFH)
Temporary real: 16383 (3FFFH)

Valori speciali

Il coprocessore matematico 80287/80387 definisce i seguenti valori speciali per incrementare la flessibilità delle elaborazioni numeriche:

- Non normali
- Non normalizzati
- Valori indefiniti
- Valori NAN (non numeri)
- Rappresentazione di $+\infty$ e di $-\infty$
- Zero con segno

4.2 Istruzioni dell'80287/80387

Gli acronimi delle istruzioni dell'80287/80387 hanno tutti il prefisso "F" o "FN".

F2XM1
(2 raised to the X power Minus 1)

Durata (cicli): 500 (80287)

Descrizione: Calcola il risultato dell'operazione 2 elevato a X meno 1

Operazione: L'istruzione F2XM1 calcola la funzione $2^X - 1$. Il valore di X è contenuto nel registro ST che si trova in cima allo stack dell'80287/80387 e il risultato viene memorizzato nello stesso registro.

Sintassi: F2XM1 (nessun operando)

Flag di eccezione: U, P

Esempio:
```
F2XM1
```

FABS
(ABSolute value)

Durata (cicli): 14 (80287)

Descrizione: Calcola il valore assoluto dell'elemento in cima allo stack dell'80287/80387

Operazione: L'istruzione FABS estrae il contenuto del registro che si trova in cima allo stack dell'80287/80387 e lo sostituisce con il corrispondente valore assoluto.

Sintassi: FABS (nessun operando)

Flag di eccezione: I

Esempio:
```
FABS
```

FADD
(ADD real)

Durata (cicli): 85, 105, 110 (80287)

Descrizione: Somma gli operandi sorgente e destinazione

Operazione: L'istruzione FADD somma gli operandi sorgente e destinazione e memorizza il risultato così ottenuto nell'operando destinazione.
Quando l'istruzione FADD non contiene operandi, viene estatto l'elemento che si trova in cima allo stack dell'80287/80387, viene sommato ad esso il secondo elemento dello stack e il risultato così ottenuto viene memorizzato in cima allo stack.
Il secondo formato dell'istruzione FADD permette all'operando sorgente di essere un numero reale allocato in memoria, con l'operando destinazione implicitamente contenuto nel registro che si trova in cima allo stack dell'80287/80387. Il risultato della somma di questi due operandi viene memorizzato in cima allo stack.
Il terzo formato dell'istruzione FADD permette di selezionare qualunque altro registro dello stack dell'80287/80387 come uno degli operandi, mentre l'altro operando rimane il contenuto del registro che si trova in cima allo stack. L'istruzione somma questi due operandi e restituisce il risultato all'operando destinazione.

Sintassi: FADD (nessun operando)
FADD *operando_sorgente*
FADD *operando_destinazione,operando_sorgente*

Flag di eccezione: I, D, U, P, O

Esempio:
```
FADD
FADD    NUM_REALE
FADD    ST,ST(1)
FADD    ST(7),ST
```

FBLD
(packed decimal Bcd LoaD)

Durata (cicli): 300 (80287)

Descrizione: Converte un numero BCD (decimale codificato in binario) nel formato temporary real e memorizza il risultato nello stack

Operazione: L'istruzione FBLD converte l'operando sorgente, che è espresso nel formato BCD compattato, nel formato temporary real e memorizza il risultato nello stack dell'80287/80387. Questa istruzione non esegue alcun controllo sulla validità del formato BCD dell'operando sorgente, ma presuppone che le cifre che lo costituiscono abbiano valori compresi tra 0H e 9H.

Sintassi: FBLD *operando_sorgente*

Flag di eccezione: I

Esempio:

```
FBLD    VALORE_BCD
```

FBSTP
(packed decimal Bcd STore and Pop)

Durata (cicli): 530 (80287)

Descrizione: Converte il contenuto del registro che si trova in cima allo stack nel formato BCD (decimale codificato in binario), memorizza il risultato nell'operando destinazione ed esegue una operazione di estrazione dell'elemento in cima allo stack.

Operazione: L'istruzione FBSTP converte il valore che è memorizzato in cima allo stack dell'80287/80387 nel formato intero decimale compattato, memorizza il risultato della conversione nell'operando destinazione ed esegue un'operazione di estrazione del contenuto della cima dello stack dell'80287/80387.

Sintassi: FBSTP *operando_destinazione*

Flag di eccezione: I

Esempio:

```
FBSTP   VALORE_BCD
```

FCHS
(CHange Sign)

Durata (cicli): 15 (80287)

Descrizione: Cambia il segno al contenuto del registro che si trova in cima allo stack dell'80287/80387

Operazione: L'istruzione FCHS complementa il bit di segno del valore numerico contenuto nel registro in cima allo stack dell'80287/80387.

Sintassi: FCHS (nessun operando)

Flag di eccezione: I

Esempio:
FCHS

FCLEX
(CLear EXception)

Durata (cicli): 5 (80287)

Descrizione: Azzera i flag di eccezioni, il flag IR e il flag B

Operazione: L'istruzione FCLEX azzera tutti i flag di eccezione, il flag di richiesta di interrupt (IR) e i flag di disponibilità del coprocessore (B), presenti nella parola di stato.

Sintassi: FCLEX (nessun operando)

Flag di eccezione: Nessuno

Esempio:
FCLEX

FCOM
(COMpare real)

Durata (cicli): 45, 65, 70 (80287)

Descrizione: Confronta la cima dello stack con l'operando sorgente

Operazione: L'istruzione FCOM confronta il contenuto del registro che si trova in cima allo stack dell'80287/80387 con l'operando sorgente. Se l'istruzione non specifica l'operando sorgente, si assume che questo sia ST(1). L'operando sorgente può essere un registro dello stack dell'80287/80387 oppure un operando in memoria espresso nel formato long real. I bit C3 e C0, all'interno della parola di stato dell'80287/80387, assumono un valore dipendente dal risultato del confronto nel modo seguente:

se ST > *operando_sorgente* allora C0 = 0 e C3 = 0

se ST < *operando_sorgente* allora C0 = 1 e C3 = 0
se ST = *operando_sorgente* allora C0 = 0 e C3 = 1
se ST < > *operando_sorgente* allora C0 = 1 e C3 = 1

Sintassi: FCOM (nessun operando)
FCOM *operando_sorgente*

Flag di eccezione: I, D

Esempio:

```
FCOM
FCOM    VAL_LONG_REAL
FCOM    VAL_SHORT_REAL
```

FCOMP
(COMpare real and Pop)

Durata (cicli): 47, 68, 72 (80287)

Descrizione: Estrae dallo stack l'elemento in cima e lo confronta con l'operando sorgente

Operazione: L'istruzione FCOMP estrae il contenuto del registro che si trova in cima allo stack dell'80287/80387 e lo confronta con l'operando sorgente. Se l'istruzione non specifica alcun operando, si assume che questo sia ST(1).
L'operando sorgente può essere un registro dello stack dell'80287/80387 oppure un operando in memoria espresso nel formato long real. I bit C3 e C0, all'interno della parola di stato dell'80287/80387, assumono un valore dipendente dal risultato del confronto nel modo seguente:

se ST > *operando_sorgente* allora C0 = 0 e C3 = 0
se ST < *operando_sorgente* allora C0 = 1 e C3 = 0
se ST = *operando_sorgente* allora C0 = 0 e C3 = 1
se ST < > *operando_sorgente* allora C0 = 1 e C3 = 1

Sintassi: FCOMP (nessun operando)
FCOMP *operando_sorgente*

Flag di eccezione: I, D

Esempio:

```
FCOMP
FCOMP   VAL_LONG_REAL
FCOMP   VAL_SHORT_REAL
```

FCOMPP
(COMpare real, Pop and Pop)

Durata (cicli): 50 (80287)

Descrizione: Confronta la cima dello stack con ST(1) ed esegue due operazioni di estrazione di elementi dallo stack

Operazione: L'istruzione FCOMPP confronta il contenuto del registro che si trova in cima allo stack dell'80287/80387 con il contenuto del registro ST(1) e poi estrae i primi due elementi che si trovano sullo stack. I bit C3 e C0, all'interno della parola di stato dell'80287/80387, assumono un valore dipendente dal risultato del confronto nel modo seguente:

se ST > *operando_sorgente* allora C0 = 0 e C3 = 0
se ST < *operando_sorgente* allora C0 = 1 e C3 = 0
se ST = *operando_sorgente* allora C0 = 0 e C3 = 1
se ST < > *operando_sorgente* allora C0 = 1 e C3 = 1

Sintassi: FCOMPP (nessun operando)

Flag di eccezione: I, D

Esempio:
```
FCOMPP
```

FCOS (80387)
(COSine)

Durata (cicli): N/A

Descrizione: Calcola il COSENO di un angolo

Operazione: L'istruzione FCOS calcola il COSENO di un angolo, il cui valore, esente da limitazioni, è memorizzato nel registro che si trova in cima allo stack dell'80287/80387. Al termine, il risultato dell'elaborazione viene memorizzato in cima allo stack.

Sintassi: FCOS (nessun operando)

Flag di eccezione: D, IS, I, P

Esempio:
```
FCOS
```

FDECSTP
(DECrement STack Pointer)

Durata (cicli): 9 (80287)

Descrizione: Sottrae una unità al contenuto del registro puntatore allo stack dell'80287/80387.

Operazione: L'istruzione FDECSTP sottrae una unità al valore del puntatore alla cima dello stack dell'80287/80387 che è memorizzato nella parola di stato.

Sintassi: FDECSTP (nessun operando)

Flag di eccezione: Nessuno

Esempio:

```
FDECSTP
```

FDISI
(DISable Interrupts)

Durata (cicli): 5 (80287)

Descrizione: Impedisce all'80287/80387 di emettere una richiesta di interruzione

Operazione: L'istruzione FDISI impedisce che l'80287/80387 esegua una richiesta di interruzione, ponendo a 1 il bit di abilitazione dell'interruzione nella parola di controllo.

Sintassi: FDISI (nessun operando)

Flag di eccezione: Nessuno

Esempio:

```
FDISI
```

FDIV
(DIVide real)

Durata (cicli): 198, 220, 225 (80287)

Descrizione: Divide l'operando destinazione per l'operando sorgente e memorizza il risultato nell'operando destinazione.

Operazione: L'istruzione FDIV divide l'operando destinazione per l'operando sorgente e restituisce il quoziente all'operando destinazione.
Se l'istruzione FDIV non specifica alcun operando, sorgente o destinazione, si assume che essi siano, rispettivamente, ST(1) e ST. L'istruzione utilizza ST come operando sorgente, ST(1) come operando destinazione, esegue un'operazione di estrazione di un elemento dallo stack e restituisce il risultato a ST.

Sintassi: FDIV (nessun operando)
FDIV *operando__sorgente*
FDIV *operando__destinazione,operando__sorgente*

Flag di eccezione: I, D, O, U, Z, P

Esempio:
```
FDIV
FDIV    ST,ST(7)
FDIV    ST(7),ST
FDIV    VAL__LONG__REAL
FDIV    VAL__SHORT__REAL
```

FDIVP
(DIVide real and Pop)

Durata (cicli): 202 (80287)

Descrizione: Divide l'operando destinazione per l'operando sorgente, esegue un'operazione di estrazione di un elemento dallo stack

Operazione: L'istruzione FDIVP considera ST come operando sorgente e ST(*n*) come operando destinazione; estrae dallo stack dell'80287/80387 l'elemento che si trova in cima, esegue la divisione e restituisce il risultato al registro ST(*n*).

Sintassi: FDIVP *operando__destinazione,operando__sorgente*

Flag di eccezione: I, D, O, U, Z, P

Esempio:
```
FDIVP   ST(7),ST
```

FDIVR
(DIVide real Reversed)

Durata (cicli): 199, 221, 226 (80287)

Descrizione: Divide l'operando sorgente per l'operando destinazione e memorizza il risultato nell'operando destinazione.

Operazione: L'istruzione FDIVR esegue l'operazione di divisione in ordine invertito rispetto all'istruzione FDIV. FDIVR divide, infatti, l'operando sorgente per l'operando destinazione e memorizza il quoziente nell'operando destinazione.
Se l'istruzione FDIVR non specifica alcun operando, destinazione o sorgente, si assume che essi siano, rispettivamente, ST(1) e ST.

Sintassi: FDIVR (nessun operando)
FDIVR *operando_sorgente*
FDIVR *operando_destinazione,operando_sorgente*

Flag di eccezione: I, D, O, U, P, Z

Esempio:
```
FDIVR
FDIVR    VAL_LONG_REAL
FDIVR    VAL_SHORT_REAL
FDIVR    ST,ST(7)
FDIVR    ST(7),ST
```

FDIVRP
(DIVide real Reversed and Pop)

Durata (cicli): 203 (80287)

Descrizione: Divide l'operando sorgente per l'operando destinazione, memorizza il risultato nell'operando destinazione ed esegue una operazione di estrazione di un elemento dallo stack.

Operazione: L'istruzione FDIVRP esegue l'operazione di divisione in ordine invertito rispetto all'istruzione FDIVP. FDIVRP divide, infatti, l'operando sorgente per l'operando destinazione e memorizza il quoziente nell'operando destinazione.
L'operando sorgente è sempre il registro ST, l'operando destinazione è ST(*n*).

Sintassi: FDIVRP *operando_destinazione,operando_sorgente*

Flag di eccezione: I, D, O, U, P, Z

Esempio:
```
FDIVRP      ST(7),ST
FDIVRP      ST(1),ST
```

FENI
(ENable Interrupts)

Durata (cicli): 5 (80287)

Descrizione: Abilita la richiesta di interruzione

Operazione: L'istruzione FENI azzera l'indicatore di abilitazione alle interruzioni nella parola di controllo, permettendo al coprocessore di generare richieste di interruzioni.

Sintassi: FENI (nessun operando)

Flag di eccezione: Nessuno

Esempio:
```
  FENI
```

FFREE
(FREE register)

Durata (cicli): 11 (80287)

Descrizione: Aggiorna il valore dell'etichetta, che è associata al registro destinazione, al valore vuoto.

Operazione: L'istruzione FFREE pone a "vuoto" nella tag word lo stato dell'indicatore relativo al registro destinazione, senza modificare il contenuto di tale registro.

Sintassi: FFREE *operando_destinazione*

Flag di eccezione: Nessuno

Esempio:
```
  FFREE     ST(1)
```

FIADD
(Integer ADD)

Durata (cicli): 120, 125 (80287)

Descrizione: Somma gli operandi sorgente e destinazione e memorizza il risultato nell'operando destinazione.

Operazione: L'istruzione FIADD somma gli operandi sorgente e destinazione e restituisce il valore somma all'operando destinazione. L'operando destinazione implicito è sempre il registro che si trova in cima allo stack dell'80287/80387, cioè ST.

Sintassi: FIADD *operando_sorgente*

Flag di eccezione: I, O, P, D

Esempio:
```
FIADD    INT_WORD
FIADD    SHORT_INT
```

FICOM
(Integer COMpare)

Durata (cicli): 80, 85 (80287)

Descrizione: Confronta la cima dello stack con l'operando sorgente

Operazione: L'istruzione FICOM converte l'operando sorgente nel formato temporary real e lo confronta con il contenuto del registro che si trova in cima allo stack dell'80287/80387. I bit C3 e C0, all'interno della parola di stato dell'80287/80387, assumono un valore dipendente dal risultato del confronto nel modo seguente:

se ST > *operando_sorgente* allora C0 = 0 e C3 = 0
se ST < *operando_sorgente* allora C0 = 1 e C3 = 0
se ST = *operando_sorgente* allora C0 = 0 e C3 = 1
se ST < > *operando_sorgente* allora C0 = 1 e C3 = 1

Sintassi: FICOM *operando_sorgente*

Flag di eccezione: I, D

Esempio:
```
FICOM    INT_WORD
FICOM    SHORT_INT
```

FICOMP
(Integer COMpare and Pop)

Descrizione: Confronta la cima dello stack con l'operando sorgente, estrae dallo stack l'elemento in cima.

Operazione: L'istruzione FICOMP è identica all'istruzione FICOM, ad eccezione dell'operazione addizionale di estrazione dallo stack dell'80287/80387 dell'elemento che si trova in cima. I bit C3 e C0, all'interno della parola di stato dell'80287/80387, assumono un valore dipendente dal risultato del confronto nel modo seguente:

se ST > *operando_sorgente* allora C0 = 0 e C3 = 0
se ST < *operando_sorgente* allora C0 = 1 e C3 = 0
se ST = *operando_sorgente* allora C0 = 0 e C3 = 1
se ST < > *operando_sorgente* allora C0 = 1 e C3 = 1

Sintassi: FICOMP *operando_sorgente*

Flag di eccezione: I, D

Esempio:

```
FICOMP   INT_WORD
FICOMP   SHORT_INT
```

FIDIV
(Integer DIVide)

Durata (cicli): 230, 236 (80287)

Descrizione: Divide l'operando destinazione per l'operando sorgente e memorizza il risultato nell'operando destinazione.

Operazione: L'istruzione FIDIV divide l'operando destinazione per l'operando sorgente e restituisce il quoziente all'operando destinazione. L'operando destinazione implicito è sempre il registro che si trova in cima allo stack dell'80287/80387, cioè ST.

Sintassi: FIDIV *operando_sorgente*

Flag di eccezione: I, D, O, U, P, Z

Esempio:

```
FIDIV    INT_WORD
FIDIV    SHORT_INT
```

FIDIVR
(Integer DIVide Reversed)

Durata (cicli): 230, 237 (80287)

Descrizione: Divide l'operando sorgente per l'operando destinazione e memorizza il risultato nell'operando destinazione.

Operazione: L'istruzione FIDIVR esegue l'operazione di divisione in ordine invertito rispetto all'istruzione FIDIV. FIDIVR divide, infatti, l'operando sorgente con l'operando destinazione implicito ST e memorizza il quoziente nel registro che si trova in cima allo stack dell'80287/80387, cioè ST.

Sintassi: FIDIVR *operando_sorgente*

Flag di eccezione: I, D, O, U, P, Z

Esempio:
```
FIDIVR   INT_WORD
FIDIVR   SHORT_INT
```

FILD
(Integer LoaD)

Durata (cicli): 50, 56, 64 (80287)

Descrizione: Converte l'operando sorgente nel formato temporary real e memorizza il risultato in cima allo stack.

Operazione: L'istruzione FILD converte l'operando sorgente dal formato intero binario al formato temporary real e memorizza il risultato nell'operando destinazione implicito, cioè in ST.

Sintassi: FILD *operando_sorgente*

Flag di eccezione: I

Esempio:
```
FILD   INT_WORD
FILD   LONG_INT
FILD   SHORT_INT
```

FIMUL
(Integer MULtiply)

Durata (cicli): 130, 136 (80287)

Descrizione: Moltiplica l'operando sorgente per l'operando destinazione e memorizza il risultato nell'operando destinazione.

Operazione: L'istruzione FIMUL moltiplica l'operando destinazione per l'operando sorgente e restituisce il risultato all'operando destinazione implicito, cioè a ST.

Sintassi: FIMUL *operando_sorgente*

Flag di eccezione: I, D, P, O

Esempio:
```
FIMUL   INT_WORD
FIMUL   SHORT_INT
```

FINCSTP
(INCrement STack Pointer)

Descrizione: Aggiunge una unità al contenuto della cima dello stack dell'80287/80387.

Operazione: L'istruzione FINCSTP aggiunge una unità al valore del puntatore alla cima dello stack dell'80287/80387 che è memorizzato nella parola di stato.

Sintassi: FINCSTP (nessun operando)

Flag di eccezione: Nessuno

Esempio:
```
FINCSTP
```

FINIT
(INITialize processor)

Durata (cicli): 5 (80287)

Descrizione: Esegue l'operazione equivalente ad un RESET a livello hardware dell'80287/80387.

Operazione: L'istruzione FINIT memorizza nella parola di controllo il valore 03FFH, azzera i flag di eccezione e svuota il contenuto dei registri dello stack dell'80287/80387. Queste operazioni equivalgono ad un RESET a livello hardware, con l'eccezione che l'istruzione FINIT non modifica la modalità di esecuzione dell'80287/80387 (modalità di indirizzamento protetta o modalità di indirizzamento reale).

Sintassi: FINIT (nessun operando)

Flag di eccezione: Nessuno

Esempio:
```
FINIT
```

FIST
(Integer STore)

Durata (cicli): 86, 88 (80287)

Descrizione: Arrotonda il valore contenuto nel registro in cima allo stack e restituisce il risultato all'operando destinazione.

Operazione: L'istruzione FIST arrotonda, in accordo con il valore del campo RC della parola di controllo, il contenuto del registro che si trova in cima allo stack dell'80287/80387 e restituisce il risultato all'operando destinazione.

Sintassi: FIST *operando_destinazione*

Flag di eccezione: I, P

Esempio:
```
FIST    INT_WORD
FIST    SHORT_INT
```

FISTP
(Integer STore and Pop)

Durata (cicli): 88, 90, 100 (80287)

Descrizione: Arrotonda il valore contenuto in cima allo stack, memorizza il risultato nell'operando destinazione ed esegue un'operazione di estrazione di un elemento dallo stack.

Operazione: L'istruzione FISTP arrotonda il contenuto del registro che si trova in cima allo stack, come accade nell'istruzione FIST, memorizzando il risultato nell'operando destinazione; inoltre, estrae dallo stack dell'80287/80387 l'elemento che si trova in cima.

Sintassi: FISTP *operando_destinazione*

Flag di eccezione: I, P

Esempio:

```
FISTP   INT_WORD
FISTP   LONG_INT
FISTP   SHORT_INT
```

FISUB
(Integer SUBtract)

Durata (cicli): 120, 125 (80287)

Descrizione: Sottrae l'operando sorgente all'operando destinazione e memorizza il risultato nell'operando destinazione

Operazione: L'istruzione FISUB sottrae l'operando sorgente all'operando destinazione implicito, cioè a ST, memorizzando la differenza nel registro che si trova in cima allo stack dell'80287/80387, cioè in ST.

Sintassi: FISUB *operando_sorgente*

Flag di eccezione: I, D, P, O

Esempio:

```
FISUB   INT_WORD
FISUB   SHORT_INT
```

FISUBR
(Integer SUBtraction Reversed)

Durata (cicli): 120, 125 (80287)

Decsrizione: Sottrae l'operando destinazione all'operando sorgente e memorizza il risultato nell'operando destinazione.

Operazione: L'istruzione FISUBR esegue l'operazione di sottrazione in ordine invertito rispetto all'istruzione FISUB. FISUBR, infatti, sottrae l'operan-

do destinazione implicito, cioè ST, all'operando sorgente e memorizza il risultato nel registro che si trova in cima allo stack dell'80287/80387, cioè in ST.

Sintassi: FISUBR *operando_sorgente*

Flag di eccezione: I, D, P, O

Esempio:

```
FISUBR   INT_WORD
FISUBR   SHORT_INT
```

FLD
(LoaD real)

Durata (cicli): 20, 43, 46, 57 (80287)

Descrizione: Memorizza l'operando sorgente in cima allo stack

Operazione: L'istruzione FLD memorizza l'operando sorgente nel registro (ST) che si trova in cima allo stack dell'80287/80387, decrementando di una unità il valore del puntatore allo stack dell'80287/80387 e copiando il contenuto dell'operando sorgente nel registro che viene referenziato dal puntatore così aggiornato.

Sintassi: FLD *operando_sorgente*

Flag di eccezione: I, D

Esempio:

```
FLD      ST(7)
FLD      LONG_REAL
FLD      SHORT_REAL
```

FLD1
(LoaD +1.0 onto top of stack)

Durata (cicli): 18 (80287)

Descrizione: Memorizza +1.0 in cima allo stack dell'80287/80387

Operazione: L'istruzione FLD1 memorizza, nel registro che si trova in cima allo stack dell'80287/80387, la costante +1.0 con una precisione, nel formato temporary real, di 64 bit e, nel formato decimale, di 19 cifre.

Sintassi: FLD1 (nessun operando)

Flag di eccezione: I

Esempio:
```
FLD1
```

FLDCW
(LoaD Control Word)

Durata (cicli): 10 (80287)

Descrizione: Memorizza nella parola di controllo l'operando sorgente

Operazione: L'istruzione FLDCW modifica il contenuto corrente della parola di controllo del microprocessore con la word definita dall'operando sorgente.

Sintassi: FLDCW *operando_sorgente*

Flag di eccezione: Nessuno

Esempio:
```
FLDCW    MEM_WORD
```

FLDENV
(LoaD ENVironment)

Durata (cicli): 40 (80287)

Descrizione: Carica l'ambiente di lavoro dell'80287/80387 con l'operando sorgente

Operazione: L'istruzione FLDENV carica l'ambiente di lavoro dell'80287/80387 con i 14 byte di memoria referenziati dall'operando sorgente. L'ambiente di lavoro consiste nell'intero stato del coprocessore, eccetto il contenuto del suo stack.

Sintassi: FLDENV *operando_sorgente*

Flag di eccezione: Nessuno

Esempio:
```
FLDENV   QUATTORDICI_BYTE
```

FLDL2E
(LoaD Log base 2 of E)

Durata (cicli): 18 (80287)

Descrizione: Memorizza log_2e in cima allo stack dell'80287/80387

Operazione: L'istruzione FLDL2E memorizza, nel registro che si trova in cima allo stack dell'80287/80387, la costante log_2e con una precisione, nel formato temporary real, di 64 bit e, nel formato decimale, di 19 cifre.

Sintassi: FLDL2E (nessun operando)

Flag di eccezione: I

Esempio:
```
FLDL2E
```

FLDL2T
(LoaD Log base 2 of Ten)

Durata (cicli): 19 (80287)

Descrizione: Memorizza $log_2 10$ in cima allo stack dell'80287/80387

Operazione: L'istruzione FLDL2T memorizza, nel registro che si trova in cima allo stack dell'80287/80387, la costante $log_2 10$ con una precisione, nel formato temporary real, di 64 bit e, nel formato decimale, di 19 cifre.

Sintassi: FLDL2T (nessun operando)

Flag di eccezione: I

Esempio:
```
FLDL2T
```

FLDLG2
(LoaD LoG_{10} of 2)

Durata (cicli): 21 (80287)

Descrizione: Memorizza $log_{10}2$ in cima allo stack dell'80287/80387

Operazione: L'istruzione FLDLG2 memorizza, nel registro che si trova in ci-

ma allo stack dell'80287/80387, la costante $\log_{10}2$ con una precisione, nel formato temporary real, di 64 bit e, nel formato decimale, di 19 cifre.

Sintassi: FLDLG2 (nessun operando)

Flag di eccezione: I

Esempio:
FLDLG2

FLDLN2
(LoaD LN of 2)

Durata (cicli): 20 (80287)

Descrizione: Memorizza $\log_e 2$ in cima allo stack dell'80287/80387

Operazione: L'istruzione FLDLN2 memorizza, nel registro che si trova in cima allo stack dell'80287/80387, la costante $\log_e 2$ con una precisione, nel formato temporary real, di 64 bit e, nel formato decimale, di 19 cifre.

Sintassi: FLDLN2 (nessun operando)

Flag di eccezione: I

Esempio:
FLDLN2

FLDPI
(LoaD PI)

Durata (cicli): 19 (80287)

Descrizione: Memorizza π in cima allo stack dell'80287/80387

Operazione: L'istruzione FLDPI memorizza, nel registro che si trova in cima allo stack dell'80287/80387, la costante π con una precisione, nel formato temporary real, di 64 bit e, nel formato decimale, di 19 cifre.

Sintassi: FLDPI (nessun operando)

Flag di eccezione: I

Esempio:
FLDPI

FLDZ
(LoaD Zero)

Durata (cicli): 14 (80287)

Descrizione: Memorizza +0.0 in cima allo stack dell'80287/80387

Operazione: L'istruzione FLDZ memorizza, nel registro che si trova in cima allo stack dell'80287/80387, la costante +0.0 con una precisione, nel formato temporary real, di 64 bit e, nel formato decimale, di 19 cifre.

Sintassi: FLDZ (nessun operando)

Flag di eccezione: I

Esempio:

```
FLDZ
```

FMUL
(MULtiply real)

Durata (cicli): 97, 118, 120, 138, 161 (80287)

Descrizione: Moltiplica l'operando destinazione per l'operando sorgente e memorizza il risultato nell'operando destinazione

Operazione: L'istruzione FMUL moltiplica l'operando destinazione per l'operando sorgente e memorizza il risultato nell'operando destinazione. Se FMUL non specifica alcun operando, destinazione o sorgente, si assume che essi siano, rispettivamente, ST(1) e ST.

Sintassi: FMUL (nessun operando)
FMUL *operando_sorgente*
FMUL *operando_destinazione,operando_sorgente*

Flag di eccezione: I, D, U, P, O

Esempio:

```
FMUL
FMUL    LONG_REAL
FMUL    SHORT_REAL
FMUL    ST,ST(7)
FMUL    ST(7),ST
```

FMULP
(MULtiply real and Pop)

Durata (cicli): 100, 142 (80287)

Descrizione: Moltiplica l'operando destinazione per l'operando sorgente, memorizza il risultato nell'operando destinazione ed esegue un'operazione di estrazione dallo stack.

Operazione: L'istruzione FMULP ha come operando sorgente il contenuto del registro che si trova in cima allo stack dell'80287/80387, cioè ST, e come operando destinazione il contenuto del registro ST(*n*). Dopo l'operazione di moltiplicazione, il coprocessore esegue un'operazione di estrazione dallo stack e memorizza il risultato nel registro ST(*n*).

Sintassi: FMULP *operando_destinazione,operando_sorgente*

Flag di eccezione: I, D, U, O, P

Esempio:
```
FMULP    ST(7),ST
```

FNCLEX
(CLear EXceptions)

Durata (cicli): 5 (80287)

Descrizione: Azzera i flag di eccezione, il bit IR e il bit B.

Operazione: L'istruzione FNCLEX azzera tutti i flag di eccezione, il bit IR (richiesta di interruzione) e il bit B (disponibilità del coprocessore) nella parola di stato.

Sintassi: FNCLEX (nessun operando)

Flag di eccezione: Nessuno

Esempio:
```
FNCLEX
```

FNDISI
(DISable Interrupts)

Durata (cicli): N/A

Descrizione: Indicatore di abilitazione alle interruzioni = 1 nella parola di controllo.

Operazione: L'istruzione FNDISI pone a 1 l'indicatore di abilitazione alle interruzioni nella parola di controllo ed impedisce che il coprocessore emetta delle richieste di interruzioni.

Sintassi: FNDISI (nessun operando)

Flag di eccezione: Nessuno

Esempio:

```
FNDISI
```

FNENI
(ENable Interrupts)

Durata (cicli): N/A

Descrizione: Indicatore di abilitazione alle interruzioni = 0

Operazione: L'istruzione FNENI azzera l'indicatore di abilitazione alle interruzioni nella parola di controllo e permette al coprocessore di generare le richieste di interruzioni.

Sintassi: FNENI (nessun operando)

Flag di eccezione: Nessuno

Esempio:

```
FNENI
```

FNINIT
(INITialize processor)

Durata (cicli): 5 (80287)

Descrizione: Esegue un'operazione di RESET del coprocessore

Operazione: L'istruzione FNINIT memorizza il valore 03FFH nella parola di controllo, azzera i flag di eccezioni, gli indicatori di interrupt e di disponibilità del coprocessore, e svuota tutti i registri dello stack dell'80287/80387. Questa operazione equivale ad un RESET a livello hardware, con l'eccezio-

ne che l'istruzione FNINIT non modifica la modalità di esecuzione dell'80287/80387 (modalità di indirizzamento protetta o modalità di indirizzamento reale).

Sintassi: FNINIT (nessun operando)

Flag di eccezione: Nessuno

Esempio:
```
FNINIT
```

FNOP
(No OPeration)

Durata (cicli): 13 (80287)

Descrizione: Nessuna operazione

Operazione: L'istruzione FNOP non esegue alcuna operazione.

Sintassi: FNOP (nessun operando)

Flag di eccezione: Nessuno

Esempio:
```
FNOP
```

FNSAVE
(SAVE state)

Durata (cicli): N/A

Descrizione: Memorizza lo stato del coprocessore nella locazione di memoria specificata dall'operando destinazione

Operazione: L'istruzione FNSAVE salva l'ambiente di lavoro dell'80287/80387, compresi anche i registri dello stack, a partire dalla locazione di memoria specificata dall'operando destinazione. L'80287/80387 viene poi inizializzato.

Sintassi: FNSAVE *operando_destinazione*

Flag di eccezione: Nessuno

Esempio:
```
FNSAVE   LOC_MEM
```

FNSTCW
(STore Control Word)

Durata (cicli): 15 (80287)

Descrizione: Memorizza il contenuto della parola di controllo nell'operando destinazione

Operazione: L'istruzione FNSTCW memorizza il contenuto corrente della parola di controllo nella locazione di memoria specificata dall'operando destinazione.

Sintassi: FNSTCW (nessun operando)

Flag di eccezione: Nessuno

Esempio:
```
FNSTCW   LOC_MEM
```

FNSTENV
(STore ENVironment)

Durata (cicli): 45 (80287)

Descrizione: Memorizza l'ambiente di lavoro dell'80287/80387 nell'operando destinazione

Operazione: L'istruzione FNSTENV memorizza l'ambiente corrente di lavoro dell'80287/80387, i puntatori di eccezioni, le parole di informazione (tag word), la parola di controllo e la parola di stato alla locazione di memoria specificata dall'operando destinazione.

Sintassi: FNSTENV *operando_destinazione*

Flag di eccezione: Nessuno

Esempio:
```
FNSTENV LOC_MEM
```

FNSTSW
(STore Status Word)

Durata (cicli): 15 (80287)

Descrizione: Memorizza la parola di stato nell'operando destinazione

Operazione: L'istruzione FNSTSW memorizza il contenuto corrente della parola di stato dell'80287/80387 nella locazione di memoria specificata dall'operando destinazione.

Sintassi: FNSTSW *operando_destinazione*

Flag di eccezione: Nessuno

Esempio:

```
FNSTSW   LOC_MEM
```

FPATAN
(Partial Arc TANgent)

Durata (cicli): 650 (80287)

Descrizione: Calcola l'ARCOTANGENTE di una funzione

Operazione: L'istruzione FPATAN estrae dal registro che si trova in cima allo stack il valore X; esegue una seconda estrazione dalla nuova cima dello stack, in cui è contenuto il valore Y; calcola la funzione ARCTAN(Y/X) e memorizza il risultato nel registro che si trova in cima allo stack dell'80287/80387, cioè in ST. I valori X e Y devono soddisfare la seguente relazione: $0 < Y < X < \infty$.

Sintassi: FPATAN (nessun operando)

Flag di eccezione: U, P

Esempio:

```
FPATAN
```

FPREM
(Partial REMainder)

Durata (cicli): 125 (80287)

Descrizione: Calcola il resto della divisione di ST per ST(1) e pone il risultato in ST.

Operazione: L'istruzione FPREM divide il contenuto corrente del registro che si trova in cima allo stack dell'80287/80387 per il contenuto dell'elemen-

to successivo ST(1) e memorizza il resto della divisione (eseguita con sottrazioni successive) nel registro in cima allo stack, cioè in ST.

Sintassi: FPREM (nessun operando)

Flag di eccezione: I, U, D

Esempio:

```
FPREM
```

FPTAN
(Partial TANgent)

Durata (cicli): 450 (80287)

Descrizione: Calcola la tangente di un angolo

Operazione: L'istruzione FPTAN calcola il valore di $Y/X = TAN(\theta)$ di un angolo. θ deve essere compreso tra 0, incluso, e $\pi/4$ ed è memorizzato nel registro che si trova in cima allo stack dell'80287/80387. Al termine dell'operazione, Y sostituisce θ nello stack e X viene memorizzato in cima allo stack. L'istruzione non segnala eventuali traboccamenti dai valori limite dell'angolo. I valori delle funzioni SIN, COS, ecc., devono essere derivati dal risultato di questa istruzione. Nota per l'80387: θ può rappresentare qualunque misura di angolo e questo coprocessore supporta le istruzioni FSIN e FCOS.

Sintassi: FPTAN (nessun operando)

Flag di eccezione: I, P

Esempio:

```
FPTAN
```

FRNDINT
(RouND to INTeger)

Durata (cicli): 45 (80287)

Descrizione: Arrotonda ad un intero il contenuto della cima dello stack dell'80287/80387

Operazione: L'istruzione FRNDINT arrotonda ad un intero il contenuto del registro che si trova in cima allo stack dell'80287/80387. Esistono quattro

modalità di arrotondamento, che si basano sul valore del campo RC della parola di controllo:

RC=00	Arrotonda all'intero più vicino: se il valore si trova esattamente tra due interi, viene scelto l'intero pari
RC=01	Arrotonda all'intero inferiore
RC=10	Arrotonda all'intero superiore
RC=11	Arrotonda verso il valore zero

Sintassi: FRNDINT (nessun operando)

Flag di eccezione: I, P

Esempio:

```
FRNDINT
```

FRSTOR
(ReSTORe state)

Durata (cicli): 205 (80287)

Descrizione: Ripristina lo stato dell'80287/80387

Operazione: L'istruzione FRSTOR ripristina lo stato dell'80287/80387 che era memorizzato nei 94 byte di memoria referenziati dall'operando sorgente. Per ragioni di compatibilità, lo stato dell'80287/80387 dove essere stato salvato tramite l'istruzione FSAVE o l'istruzione FNSAVE.

Sintassi: FRSTOR *operando_sorgente*

Flag di eccezione: Nessuno

Esempio:

```
FRSTOR   LOC_MEM
```

FSAVE
(SAVE state)

Durata (cicli): 205 (80287)

Descrizione: Salva lo stato dell'80287/80387

Operazione: L'istruzione FSAVE memorizza lo stato dell'80287/80387, tra cui

anche lo stack di registri e l'ambiente di lavoro, alla locazione di memoria referenziata dall'operando destinazione, Poi, l'80287/80387 viene inizializzato.

Sintassi: FSAVE *operando_destinazione*

Flag di eccezione: Nessuno

Esempio:

```
FSAVE   LOC_MEM
```

FSCALE

(SCALE by power of 2)

Durata (cicli): 35 (80287)

Descrizione: Moltiplica o divide per potenze di 2

Operazione: L'istruzione FSCALE considera il valore che è memorizzato nel registro ST(1), lo interpreta come un intero, e lo somma all'esponente del numero che è memorizzato in ST. Il risultato è $ST \times 2^{ST(1)}$.

Sintassi: FSCALE (nessun operando)

Flag di eccezione: I, U, O

Esempio:

```
FSCALE
```

FSETPM

(SET Protected Mode)

Durata (cicli): 5 (80287)

Descrizione: Abilita la modalità di indirizzamento protetta

Operazione: L'istruzione FSETPM abilita la modalità di indirizzamento protetta dell'80287/80387. Una volta che l'istruzione FSETPM è stata eseguita, l'80287/80387 rimane in modalità protetta fino alla successiva operazione di RESET a livello hardware, anche dopo che sono state eseguite le istruzioni FRSTOR, FSAVE o FINIT.

Sintassi: FSETPM (nessun operando)

Flag di eccezione: Nessuno

Esempio:
FSETPM

FSIN (80387)
(SINe)

Durata (cicli): N/A

Descrizione: Calcola il SENO di un angolo

Operazione: L'istruzione FSIN calcola il SENO di un angolo θ che è memorizzato nel registro che si trova in cima allo stack. θ non ha limitazioni di valori. Dopo la conclusione dell'operazione, il risultato viene memorizzato nel registro che si trova in cima allo stack dove era allocato il valore dell'angolo θ.

Sintassi: FSIN (nessun operando)

Flag di eccezione: D, IS, I, P

Esempio:
FSIN

FSINCOS
(SINe and COSine)

Durata (cicli): N/A

Descrizione: Calcola simultaneamente il SENO e il COSENO di un angolo

Operazione: L'istruzione FSINCOS calcola sia il SENO che il COSENO di un angolo θ, il cui valore, non soggetto ad alcuna limitazione, è memorizzato nel registro che si trova in cima allo stack dell'80287/80387. Al termine dell'operazione, il valore del COSENO viene memorizzato nel registro ST e il valore del SENO viene memorizzato nel registro ST(1).

Sintassi: FSINCOS (nessun operando)

Flag di eccezione: D, IS, I, P

Esempio:
FSINCOS

FSQRT
(SQuare RooT)

Durata (cicli): 183 (80287)

Descrizione: Alloca la radice quadrata di ST in cima allo stack

Operazione: L'istruzione FSQRT esegue la radice quadrata del contenuto del registro che si trova in cima allo stack dell'80287, cioè in ST, e memorizza il valore ottenuto in cima allo stack.

Sintassi: FSQRT (nessun operando)

Flag di eccezione: I, P, D

Esempio:

```
FSQRT
```

FST
(STore real)

Durata (cicli): 18, 87, 100 (80287)

Descrizione: Memorizza il valore contenuto in ST nell'operando destinazione

Operazione: L'istruzione FST memorizza il contenuto del registro che si trova in cima allo stack dell'80287, cioè in ST, nella locazione di memoria specificata dall'operando destinazione.

Sintassi: FST *operando_destinazione*

Flag di eccezione: I, U, O, P

Esempio:

```
FST     ST(7)
FST     VAL_LONG_REAL
FST     VAL_SHORT_REAL
```

FSTCW
(STore Control Word)

Durata (cicli): 15 (80287)

Descrizione: Memorizza il contenuto della parola di controllo nell'operando destinazione

Operazione: L'istruzione FSTCW memorizza il contenuto corrente della parola di controllo dell'80287/80387 nella locazione di memoria specificata dall'operando destinazione.

Sintassi: FSTCW *operando_destinazione*

Flag di eccezione: Nessuno

Esempio:

```
FSTCW   LOC_MEM
```

FSTENV

(STore ENVironment)

Durata (cicli): 45 (80287)

Descrizione: Memorizza lo stato dell'80287/80387 nell'operando destinazione

Operazione: L'istruzione FSTENV memorizza l'ambiente di lavoro dell'80287/80387, tra cui la parola di stato, la parola di controllo, la parola degli indicatori (tag word) e i puntatori di eccezione, nella locazione di memoria specificata dall'operando destinazione.

Sintassi: FSTENV *operando_destinazione*

Flag di eccezione: Nessuno

Esempio:

```
FSTENV  LOC_MEM
```

FSTP

(STore real and Pop)

Durata (cicli): 20, 55, 89, 102 (80287)

Descrizione: Memorizza ST in memoria ed esegue una operazione di estrazione dallo stack.

Operazione: L'istruzione FSTP memorizza il contenuto del registro che si trova in cima allo stack dell'80287/80387, cioè in ST, nella locazione di memoria specificata dall'operando destinazione, ed esegue un'operazione di estrazione dell'elemento in cima allo stack.

Sintassi: FSTP *operando_destinazione*

Flag di eccezione: I, U, O, P

Esempio:
```
FSTP    ST(2)
FSTP    VAL_LONG_REAL
FSTP    VAL_SHORT_REAL
```

FSTSW
(STore Status Word)

Durata (cicli): 15 (80287)

Descrizione: Alloca in memoria la parola di stato

Operazione: L'istruzione FSTSW memorizza il contenuto corrente della parola di stato dell'80287/80387 nella locazione di memoria specificata dall'operando destinazione.

Sintassi: FSTSW *operando_destinazione*

Flag di eccezione: Nessuno

Esempio:
```
FSTSW
```

FSUB
(SUBtract real)

Durata (cicli): 85, 105, 110 (80287)

Descrizione: Sottrae l'operando sorgente dall'operando destinazione e pone la differenza nell'operando destinazione

Operazione: L'istruzione FSUB sottrae l'operando sorgente all'operando destinazione e restituisce il risultato all'operando destinazione.
Se FSUB viene utilizzata senza operandi, si assume che essi siano, rispettivamente, ST(1) e ST. Se, invece, è specificato nell'istruzione l'operando sorgente, si assume come operando destinazione ST.

Sintassi: FSUB (nessun operando)
FSUB *operando_sorgente*
FSUB *operando_destinazione,operando_sorgente*

Flag di eccezione: I, D, P, O, U

Esempio:

```
FSUB
FSUB      VAL_LONG_REAL
FSUB      VAL_SHORT_REAL
FSUB      ST,ST(7)
FSUB      ST(7),ST
```

FSUBP

(SUBtract real and Pop)

Durata (cicli): 90

Descrizione: Sottrae l'operando sorgente all'operando destinazione, memorizza il risultato nell'operando destinazione ed esegue un'operazione di estrazione dallo stack.

Operazione: L'istruzione FSUBP sottrae l'operando sorgente all'operando destinazione e restituisce la differenza all'operando destinazione. Viene estratto, inoltre, un elemento dallo stack dell'80287/80387.

Sintassi: FSUBP *operando_destinazione,operando_sorgente*

Flag di eccezione: I, D, U, O, P

Esempio:

```
FSUBP     ST(7),ST
```

FSUBR

(SUBtract real Reversed)

Durata (cicli): 87, 105, 110 (80287)

Descrizione: Sottrae l'operando destinazione all'operando sorgente e memorizza il risultato nell'operando destinazione.

Operazione: L'istruzione FSUBR esegue l'operazione di sottrazione in ordine inverso rispetto all'istruzione FSUB. FSUBR, infatti, sottrae l'operando destinazione all'operando sorgente e restituisce la differenza all'operando destinazione.
Se l'istruzione FSUBR viene utilizzata senza operandi, destinazione o sor-

gente, si assume che essi siano, rispettivamente, ST(1) e ST. Se, invece, viene specificato solo l'operando sorgente, l'operando destinazione risulta ST.

Sintassi: FSUBR (nessun operando)
FSUBR *operando_sorgente*
FSUBR *operando_destinazione,operando_sorgente*

Flag di eccezione: I, O, D, U, P

Esempio:
```
FSUBR
FSUBR   VAL_LONG_REAL
FSUBR   VAL_SHORT_REAL
FSUBR   ST,ST(7)
FSUBR   ST(7),ST
```

FSUBRP
(SUBtract real Reversed and Pop)

Durata (cicli): 90 (80287)

Descrizione: Sottrae l'operando destinazione all'operando sorgente, memorizza ed esegue un'operazione di estrazione dallo stack

Operazione: L'istruzione FSUBRP esegue l'operazione di sottrazione in ordine inverso rispetto all'istruzione FSUBP. FSUBRP, infatti, sottrae l'operando destinazione ST(*n*) all'operando sorgente ST, estrae la cima dello stack e memorizza il risultato nell'operando destinazione ST(*n*).

Sintassi: FSUBRP *operando_destinazione,operando_sorgente*

Flag di eccezione: I, O, U, D, P

Esempio:
```
FSUBRP
```

FTST
(TeST)

Durata (cicli): 42 (80287)

Descrizione: Confronta la cima dello stack dell'80287/80387 con +0.0 e modifica di conseguenza i bit condizionali.

Operazione: L'istruzione FTST confronta il contenuto del registro che si trova in cima allo stack dell'80287/80387, cioè in ST, con la costante +0.0. Il risultato modifica di conseguenza il contenuto dei bit C3 e C0, all'interno della parola di stato, in accordo con le seguenti condizioni:

ST è positivo e non è zero	C0=0 e C3=0
ST è negativo e non è zero	C0=1 e C3=0
ST è zero	C0=0 e C3=1
ST non è un numero	C0=1 e C3=1

Sintassi: FTST (nessun operando)

Flag di eccezione: I, D

Esempio:

```
FTST
```

FWAIT
(WAIT – istruzione di CPU)

Durata (cicli): 3 + 5*n* (80287)

Descrizione: L'80286 aspetta che l'80287/80387 concluda il suo operato

Operazione: L'istruzione FWAIT sospende l'esecuzione dell'80286 fino a quando l'80287/80387 ha completato l'esecuzione dell'istruzione corrente. *n* indica il numero di volte che la CPU controlla la linea BUSY negata, prima di proseguire nell'esecuzione del programma.

Sintassi: FWAIT (nessun operando)

Flag di eccezione: Nessuno

Esempio:

```
FWAIT
```

FXAM
(eXAMine)

Durata (cicli): 17 (80287)

Descrizione: Memorizza nei bit di condizione l'esito dell'esame dello stack

Operazione: L'istruzione FXAM esamina il contenuto del registro che si trova in cima allo stack dell'80287/80387, cioè in ST, e memorizza nei bit C3, C2, C1 e C0 l'esito del controllo, in accordo con le seguenti condizioni:

C0	**C1**	**C2**	**C3**	**Risultato**
0	0	0	0	+Non normale
0	0	0	1	+No numero
0	0	1	0	−Anormale
0	0	1	1	−No numero
0	1	0	0	+Normale
0	1	0	1	+00
0	1	1	0	−Normale
0	1	1	1	−00
1	0	0	0	+0
1	0	0	1	Vuoto
1	0	1	0	−0
1	0	1	1	Vuoto
1	1	0	0	+Non normalizzato
1	1	0	1	Vuoto
1	1	1	0	−Non normalizzato
1	1	1	1	Vuoto

Sintassi: FXAM (nessun operando)

Flag di eccezione: Nessuno

Esempio:

```
FXAM
```

FXCH
(eXCHange registers)

Durata (cicli): 12 (80287)

Descrizione: Sostituisce tra loro l'operando destinazione e la cima dello stack

Operazione: L'istruzione FXCH sostituisce tra loro l'operando destinazione e il registro che si trova in cima allo stack dell'80287/80387, ST. Se l'istruzione viene utilizzata senza un operando, l'operando destinazione è per default ST(1).

Sintassi: FXCH (nessun operando)
FXCH *operando_destinazione*

Flag di eccezione: I

Esempio:

```
FXCH
FXCH   ST(7)
```

FXTRACT

(eXTRACT exponent and significand)

Durata (cicli): 50 (80287)

Descrizione: Decompone l'elemento in cima allo stack in due numeri che rappresentano i campi mantissa ed esponente.

Operazione: L'istruzione FXTRACT fattorizza il numero contenuto nel registro che si trova in cima allo stack dell'80287/80387, cioè in ST, in due numeri reali che costituiscono i campi mantissa ed esponente. L'esponente sostituisce l'operando iniziale e la mantissa diventa il contenuto del nuovo registro che viene inserito in cima allo stack, cioè ST.

Sintassi: FXTRACT (nessun operando)

Flag di eccezione: I

Esempio:

```
FXTRACT
```

FYL2X

(Y × Log base 2 of X)

Durata (cicli): 950 (80287)

Descrizione: Calcola $Y \times \log_2 X$

Operazione: L'istruzione FYL2X estrae il valore X dal registro che si trova in cima allo stack dell'80287/80387, cioè ST, e il valore Y dal registro ST(1) e memorizza il risultato dell'operazione sopra indicata in cima allo stack, cioè nel registro ST.

Sintassi: FYL2X (nessun operando)

Flag di eccezione: P

Esempio:

```
FYL2X
```

FYL2XP1
(Y × Log base 2 of X Plus 1)

Durata (cicli): 850 (80287)

Descrizione: Calcola $Y \times \log_2(X+1)$

Operazione: L'istruzione FYL2XP1 estrae il valore X dal registro che si trova in cima allo stack dell'80287/80387, cioè ST, e il valore Y dal registro ST(1) e memorizza il risultato dell'operazione sopra indicata in cima allo stack, cioè nel registro ST. Questa istruzione viene utilizzata quando il programmatore deve calcolare il logaritmo di un numero di valore prossimo a 1.

Sintassi: FYL2XP1 (nessun operando)

Flag di eccezione: P

Esempio:

```
FYL2XP1
```

5

Tecniche elementari di programmazione

Nei primi quattro capitoli del libro abbiamo trattato le caratteristiche fondamentali del linguaggio assemblatore e abbiamo esaminato i set di istruzioni dei microprocessori 80286/80386 e dei coprocessori 80287/80387. Per ognuna di queste istruzioni, i Capitoli 3 e 4 presentano un semplice esempio che ne illustra il formato e la sintassi corretta. Nessuno di questi esempi, comunque, costituisce un programma eseguibile.

Questo capitolo esamina in dettaglio la sintassi del linguaggio assemblatore e contiene numerosi esempi di programmi. Questi programmi, la cui esecuzione è esente da errori, presentano una difficoltà crescente e sono stati pensati opportunamente per facilitare un apprendimento graduale delle tecniche di programmazione.

Nel Capitolo 2 abbiamo già parlato delle istruzioni di definizione dell'ambiente di lavoro – cioè di quelle istruzioni che sono parte integrante di ogni programma in linguaggio assemblatore – e delle differenze esistenti tra i file .COM e i file .EXE.

Quasi tutti i programmi presenti nel libro possono essere contenuti in un unico segmento di codice (64 kB) e generare così una versione eseguibile di formato .COM, ma, poiché la maggior parte degli assembler supporta file di tipo .EXE, tutti gli esempi presentati – a meno di esplicita controindicazione – sono stati strutturati in modo tale da generare una versione eseguibile di formato .EXE.

L'esempio qui di seguito illustrato evidenzia l'insieme di istruzioni che sono necessarie per definire l'ambiente di lavoro e che fanno parte di ogni programma scritto in linguaggio assemblatore.

```
;(commento che descrive lo scopo del programma)

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
(qui vengono definiti i dati del programma)
DATA      ENDS

CODICE    SEGMENT PARA 'CODICE'      ;defininisce il segmento di codice
PROCEDURA PROC    FAR                ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS                 ;salva DS sullo stack
          SUB     AX,AX              ;azzera AX
          PUSH    AX                 ;salva 0 sullo stack
          MOV     AX,DATI            ;indirizzo di DATI in AX
          MOV     DS,AX              ;indirizzo di DATI in DS

(qui viene inserito il codice del programma)

          RET                        ;il controllo ritorna al DOS
PROCEDURA ENDP                       ;fine della procedura
CODICE    ENDS                       ;fine del segmento di codice

          END                        ;fine del programma
```

I primi quindici esempi di questo capitolo sono programmi di tipo generale che possono essere eseguiti su quasi ogni calcolatore che dispone di un microprocessore della famiglia Intel 8086 (dall'8088 fino all'80286 e 80386). I successivi programmi, invece, dipendono dal tipo di macchina su cui vengono eseguiti (come ad esempio i programmi scritti per il calcolatore IBM AT che utilizzano particolari routine DOS e BIOS private della libreria IBM) e, se si lavora con un calcolatore 80286 compatibile, può essere necessario apportare alcune modifiche ai programmi stessi, in base al grado di compatibilità esistente.
Questo capitolo inoltre presenta alcuni programmi che possono essere eseguiti solo su macchine dotate del microprocessore 80386.

5.1 Programmi matematici

Questo paragrafo contiene nove esempi di programmi matematici scritti in linguaggio assemblatore che realizzano alcune operazioni aritmetiche di base come l'addizione, la sottrazione, la moltiplicazione e la divisione, utilizzando differenti tecniche di programmazione.
L'ultimo esempio è la codifica di un semplice algoritmo che calcola la radice quadrata di un numero intero espresso nella notazione esadecimale. A meno di eplicita controindicazione, le operazioni aritmetiche – nel linguaggio assemblatore – vengono eseguite nel formato esadecimale, per cui è como-

do disporre di un calcolatore tascabile che aiuti il programmatore a convertire i numeri da una base numerica a un'altra per la verifica dei propri programmi.

SOMMA ESADECIMALE CON INDIRIZZAMENTO IMMEDIATO

Forse il più semplice esempio di programma nel linguaggio assemblatore è quello che esegue la somma di numeri esadecimali.
Il seguente programma – in aggiunta alle consuete istruzioni di definizione dell'ambiente di lavoro – contiene solo tre linee di codice, che sono però sufficienti a introdurre i primi concetti fondamentali della programmazione. La Figura 5.1 mostra il diagramma di flusso di questi tre passi di programma.

```
;per macchine 8088/80386
;programma che realizza una semplice operazione di somma
;esadecimale con indirizzamento immediato

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
STACK     ENDS

CODICE    SEGMENT PARA 'CODICE'      ;definisce il segmento di codice
PROCEDURA PROC    FAR                ;inizio della procedura
          ASSUME  CS:CODICE,SS:STACK
          PUSH    DS                 ;salva DS sullo stack
          SUB     AX,AX              ;azzera AX
          PUSH    AX                 ;salva 0 sullo stack

;somma di tre numeri utilizzando l'indirizzamento immediato
          MOV     AL,23H             ;il numero esad. 23H è in AL
          ADD     AL,0AH             ;somma 0AH ad AL
          ADD     AL,10H             ;somma 10H ad AL
;fine dell'esempio di somma, il risultato è in AL

          RET                        ;il controllo ritorna al DOS
PROCEDURA ENDP                       ;fine della procedura
CODICE    ENDS                       ;fine del segmento di codice

          END                        ;fine del programma
```

Prima di esaminare in dettaglio le istruzioni, parliamo della struttura del programma.
Il codice sorgente inizia con tre linee di commento che specificano il tipo di macchina su cui può essere eseguito il programma e le funzioni che il programma stesso realizza. Questa descrizione indica agli utenti quale sia lo scopo del programma e costituisce anche un promemoria per il futuro. Si tenga presente che è possibile inserire un commento in uno qualsiasi dei quattro campi di cui si compone un'istruzione, purché il commento stesso venga preceduto da un punto e virgola.

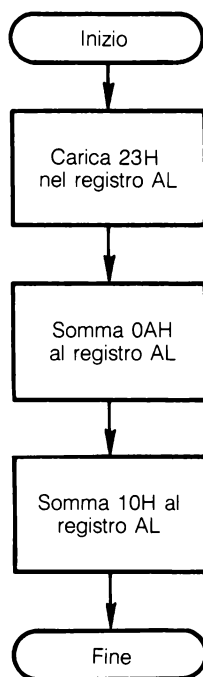


Figura 5.1 Passi del programma che realizza l'operazione di somma esadecimale

Le tre istruzioni successive definiscono il segmento di stack (la scelta dell'identificatore di segmento è arbitraria). Per favorire l'accesso al segmento di stack in fase di correzione del programma (ad esempio con il comando DEBUG), in questo segmento è stata memorizzata la stringa 'STACK '.
In questo esempio, non viene utilizzato alcun dato appartenente ad altri segmenti, per cui è superfluo definire separatamente un segmento dati. Il segmento di codice CODICE (anche qui la scelta dell'identificatore di segmento è arbitraria, purché si tratti di una stringa di caratteri) contiene la dichiarazione della procedura PROCEDURA.
Le rimanenti istruzioni (di cui il Capitolo 2 fornisce una spiegazione dettagliata) definiscono l'ambiente di lavoro e sono presenti in ogni programma scritto in linguaggio assemblatore.
Le seguenti linee di codice costituiscono il corpo del programma e realizzano l'operazione di somma esadecimale:

```
MOV     AL,23H    ;il numero esad. 23H è in AL
ADD     AL,0AH    ;somma 0AH ad AL
ADD     AL,10H    ;somma 10H ad AL
```

In questo esempio, il numero esadecimale 23H viene trasferito nel registro AL. Questo registro a 8 bit contiene il byte meno significativo del registro AX (gli otto bit più significativi di AX sono invece memorizzati nel registro AH), per cui è in grado di memorizzare un numero esadecimale compreso tra 00H e 0FFH.
MOV costituisce uno dei mnemonici più frequentemente utilizzati nella programmazione in linguaggio assemblatore 8088/80386, poiché permette di trasferire l'informazione dalla CPU alla memoria e viceversa. L'istruzione ADD AL,0AH somma il numero 0AH al contenuto corrente del registro AL e memorizza il risultato (2DH) sempre nel registro AL, mentre l'istruzione ADD AL,10H somma 10H al contenuto corrente del registro AL e memorizza il risultato finale (3DH) in AL.
Questo esempio presenta alcune limitazioni:

1. il programma non tiene conto dell'overflow se la somma memorizzata nel registro AL supera il valore 0FFH;
2. il programma non consente di sommare numeri maggiori di 0FFH;
3. il programma diventa estremamente lungo se i numeri da sommare sono ad esempio 1000;
4. se il numero cambia, il programmatore è costretto a modificare ogni linea di codice.

Anche con queste limitazioni, comunque, il programma ha una sua utilità, in quanto evidenzia una tecnica molto usuale con cui è possibile realizzare la somma di numeri esadecimali. Non dimenticate che, per imparare a programmare seriamente, dovete conoscere perfettamente quali siano i limiti dei programmi che state utilizzando.

SOTTRAZIONE ESADECIMALE CON INDIRIZZAMENTO DIRETTO

Scrivere un programma nel linguaggio assemblatore che permetta di sottrarre un numero predefinito di valori esadecimali non costituisce un compito arduo.
La Figura 5.2 mostra il semplice diagramma di flusso di un programma che sottrae tre numeri esadecimali.
Nel precedente esempio, abbiamo utilizzato il modo di indirizzamento immediato, per cui i numeri da sommare venivano direttamente memorizzati nei registri. Ora, invece, i numeri da sottrarre vengono prima memorizzati in un segmento dati, successivamente vengono referenziati tramite il loro identificatore simbolico e infine vengono sottratti utilizzando la tecnica di indirizzamento diretto.
Il seguente esempio utilizza i registri a 16 bit che sono disponibili nei microprocessori 8088/80286 (i registri generali dell'80386 sono invece a 32 bit), per cui viene incrementato il campo di valori che è possibile manipolare rispetto ai microprocessori dotati di registri a 8 bit. I registri a 16 bit infatti pos-

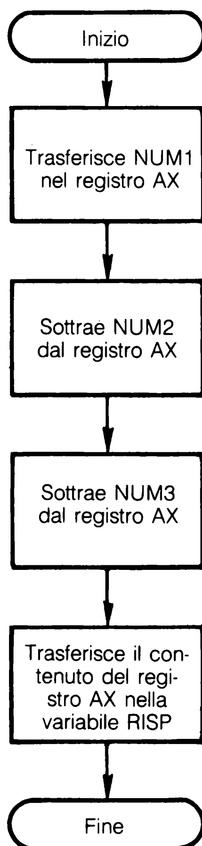


Figura 5.2 Passi del programma che realizza l'operazione di sottrazione esadecimale

sono memorizzare valori numerici compresi tra 0000H (0) e 0FFFFH (65535), mentre quelli a 8 bit sono in grado di trattare solo valori numerici compresi tra 00H (0) e 0FFH (255).

```
;per macchine 8088/80386
;programma che realizza una semplice operazione di sottrazione
;esadecimale con indirizzamento diretto

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK   ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
NUM1      DW      1234H              ;primo numero di 16 bit
NUM2      DW      24H                ;secondo numero di 16 bit
NUM3      DW      0ABCH              ;terzo numero di 16 bit
```

```
RISP      DW      ?                     ;locazione per la risposta
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'         ;definisce il segmento di codice
PROCEDURA PROC    FAR                   ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS                    ;salva DS sullo stack
          SUB     AX,AX                 ;azzera AX
          PUSH    AX                    ;salva 0 sullo stack
          MOV     AX,DATI               ;indirizzo di DATI in AX
          MOV     DS,AX                 ;indirizzo di DATI in DS

;sottrazione di tre numeri di 16 bit
          MOV     AX,NUM1               ;carica NUM1 nel registro AX
          SUB     AX,NUM2               ;sottrae NUM2 a NUM1
          SUB     AX,NUM3               ;sottrae NUM3 al risultato precedente
          MOV     RISP,AX               ;memorizza il risultato in RISP
;fine dell'esempio di sottrazione

          RET                           ;il controllo ritorna al DOS
PROCEDURA ENDP                          ;fine della procedura
CODICE    ENDS                          ;fine del segmento di codice

          END                           ;fine del programma
```

Questo è il primo esempio di programma in cui viene definito un segmento dati, che riportiamo qui di seguito:

```
DATI      SEGMENT PARA 'DATI'
NUM1      DW      1234H         ;primo numero di 16 bit
NUM2      DW      24H           ;secondo numero di 16 bit
NUM3      DW      0ABCH         ;terzo numero di 16 bit
RISP      DW      ?             ;locazione per la risposta
DATI      ENDS
```

La scelta dell'identificatore del segmento dati è arbitraria, ma si consiglia di utilizzare nomi brevi, facili da ricordare e da scrivere.
La prima linea della dichiarazione (DATI SEGMENT PARA 'DATI') indica che il segmento di nome DATI deve iniziare in memoria in corrispondenza di un paragrafo e costituisce un segmento dati, che contiene quattro variabili: NUM1, NUM2 e NUM3 rappresentano i tre numeri da sottrarre, a cui gli assembler 8088/80386 riservano una word di memoria (Define Word), cioè 16 bit; queste variabili vengono inizializzate, rispettivamente, ai valori 1234H, 24H e 0ABCH. La quarta variabile – RISP – viene dichiarata, ma non inizializzata utilizzando l'operatore ? (punto di domanda), che riserva 16 bit di memoria in cui il programma memorizzerà il risultato della sottrazione. Infine, l'ultima linea (DATI ENDS) conclude la dichiarazione del segmento dati.
Il corpo del programma che realizza l'operazione di sottrazione risulta di facile comprensione:

```
MOV       AX,NUM1   ;carica NUM1 nel registro AX
SUB       AX,NUM2   ;sottrae NUM2 a NUM1
```

```
SUB     AX,NUM3  ;sottrae NUM3 al risultato precedente
MOV     RISP,AX  ;memorizza il risultato in RISP
```

NUM1 viene trasferito nel registro AX, mentre le due linee di codice successive sottraggono rispettivamente NUM2 e NUM3 al contenuto corrente del registro AX. Infine, la quarta linea di codice trasferisce il contenuto del registro AX (754H) nella variabile RISP.
Per verificare il valore del risultato è necessario esaminare l'immagine in memoria del segmento dati utilizzando il programma DEBUG, che viene fornito dalla IBM. Si tenga presente che i numeri di tipo word occupano due byte di memoria.

```
-D DS:0000
1C9D:0000   34 12 24 00 BC 0A 54 07-00 00 00 00 00 00 00 00   4.$...T.........
1C9D:0010   4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20   MYSTACK MYSTACK
1C9D:0020   4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20   MYSTACK MYSTACK
1C9D:0030   4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20   MYSTACK MYSTACK
1C9D:0040   4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20   MYSTACK MYSTACK
1C9D:0050   4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20   MYSTACK MYSTACK
1C9D:0060   4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20   MYSTACK MYSTACK
1C9D:0070   4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20   MYSTACK MYSTACK
```

Questo esempio utilizza, in particolare, alcune soluzioni programmative che annullano le limitazioni riscontrate nel precedente esempio. Innanzitutto, l'uso di un segmento dati permette di modificare il valore dei numeri da sottrarre senza dover alterare il codice del programma; secondariamente, aumenta l'insieme dei valori trattabili, in quanto ogni numero incrementa la propria dimensione da 8 a 16 bit. In questo modo, si ottiene una maggiore precisione di calcolo e viene data al programmatore l'opportunità di operare con un campo di valori numerici più esteso.

SOMMA IN PRECISIONE MULTIPLA CON INDIRIZZAMENTO DIRETTO

I registri generali (AX, BX, CX e DX) della famiglia 8088/80286 sono limitati a 16 bit, per cui, se non ci fosse una tecnica di programmazione che consentisse di ignorare questa limitazione, i programmatori potrebbero operare solo con numeri interi compresi tra 0000H (0) e 0FFFFH (65535). Il prossimo esempio mostra come sia possibile utilizzare l'aritmetica in multipla precisione a 32 bit per manipolare – su macchine 8088/80286 – numeri fino ad un valore di 0FFFFFFFFH (4294967295). Il microprocessore 80386 supporta a livello hardware l'aritmetica a 32 bit, per cui il ricorso alla multipla precisione è necessario solo quando si devono trattare numeri superiori a 0FFFFFFFFH.
Tutti i microprocessori utilizzano il bit Carry o il bit Overflow quando eseguono una operazione aritmetica. I due programmi che abbiamo presentato, invece, ignorano le informazioni contenute in questi bit, per cui, se le

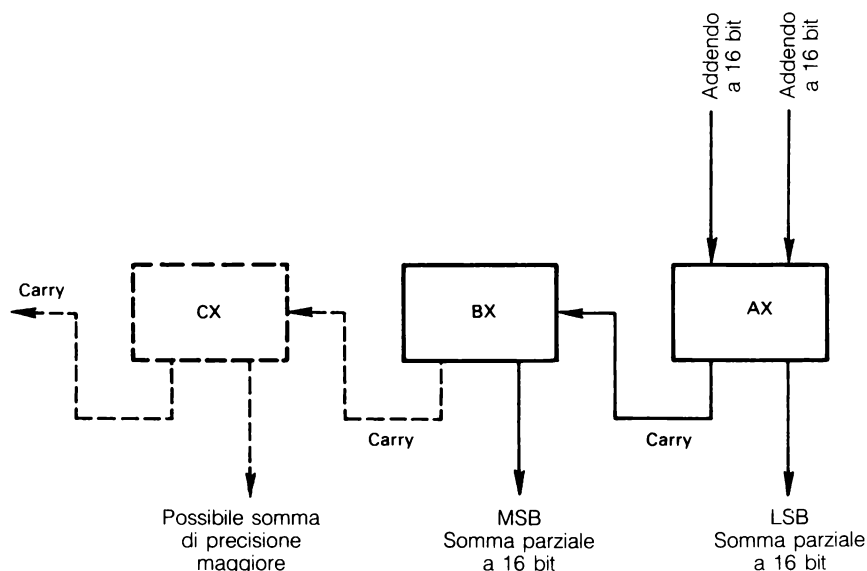


Figura 5.3 Somma in precisione multipla

operazioni generano un riporto o richiedono un prestito, i programmi stessi producono risultati errati. L'istruzione ADC (add with carry: somma con riporto) permette al programma di utilizzare l'informazione contenuta nel bit Carry e adottare così l'aritmetica in multipla precisione. La Figura 5.3 mostra come viene realizzata una operazione di somma in multipla precisione, mentre la Figura 5.4 illustra il diagramma di flusso del programma che effettua l'operazione stessa.
Si tenga presente che il bit Carry viene aggiornato a seguito di ogni operazione ADD o ADC, ma viene utilizzato solo dall'istruzione ADC.
La somma di due numeri viene realizzata partendo dal bit meno significativo (Least Significant Bit: LSB) e procedendo verso il bit più significativo (Most Significant Bit: MSB) e può produrre o meno un riporto. Sommando 1234H con 1532H, ad esempio, non viene prodotto alcun riporto, come invece accade nella somma di 0ABCDH con 5600H.
Per sommare la coppia di byte meno significativi dei due addendi si utilizza l'istruzione ADD, in quanto non esiste un riporto che sia stato generato da somme precedenti. Se questa somma produce un riporto, il bit Carry assume il valore 1 e viene sommato dall'istruzione ADC alla coppia di byte che occupano la posizione adiacente più significativa rispetto a quella corrente.
Nel prossimo esempio di programma, nei 16 bit più significativi del risultato viene accumulata solo l'informazione relativa al riporto generato, in quanto

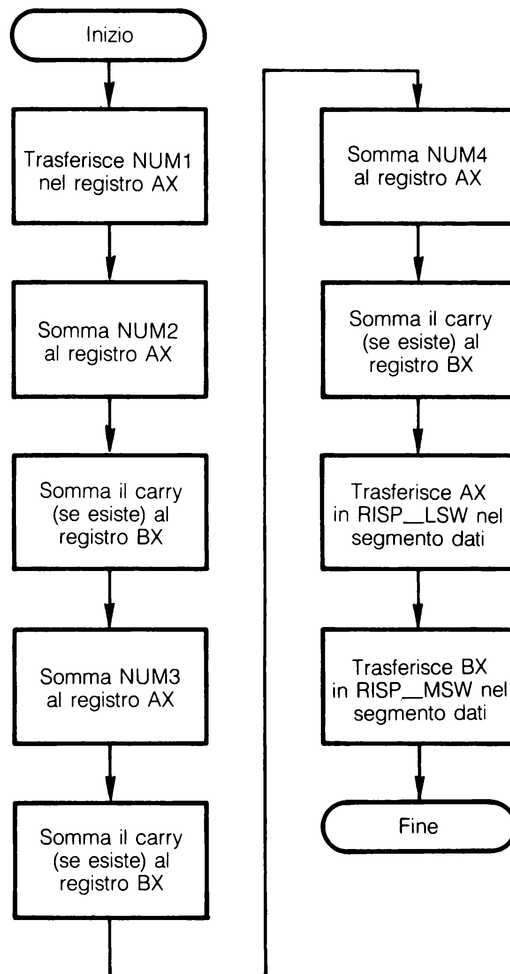


Figura 5.4 Passi di programma per la somma in precisione multipla

la dimensione di ogni addendo è 16 bit. Le istruzioni ADC hanno quindi come operando 00H ed eseguono in modalità di indirizzamento immediato. La Figura 5.5 mostra il programma completo che realizza la somma di quattro numeri di 16 bit e memorizza il risultato a 32 bit in multipla precisione in due variabili di 16 bit (RISP_LSW e RISP_MSW).

```
;per macchine 8088/80386
;programma che realizza la somma in multipla precisione con
;indirizzamento diretto

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
NUM1      DW      1234H
NUM2      DW      5678H
NUM3      DW      9ABCH
NUM4      DW      0DEF0H
RISP_LSW  DW      ?
RISP_MSW  DW      ?
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'    ;definisce il segmento di codice
PROCEDURA PROC    FAR              ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS               ;salva DS sullo stack
          SUB     AX,AX            ;azzera AX
          PUSH    AX               ;salva 0 sullo stack
          MOV     AX,DATI          ;indirizzo di DATI in AX
          MOV     DS,AX            ;indirizzo di DATI in DS

;somma in multipla precisione con indirizzamento diretto
          MOV     BX,00H           ;inizializza BX
          MOV     AX,NUM1          ;primo numero nel registro AX
          ADD     AX,NUM2          ;somma il secondo numero al primo
          ADC     BX,00H           ;se il Carry è a 1,viene sommato a BX
          ADD     AX,NUM3          ;somma il terzo numero al registro AX
          ADC     BX,00H           ;se il Carry è a 1,viene sommato a BX
          ADD     AX,NUM4          ;somma il quarto num. al registro AX
          ADC     BX,00H           ;se il Carry è a 1 viene sommato a BX
          MOV     RISP_LSW,AX      ;la somma AX è in RISP_LSW
          MOV     RISP_MSW,BX      ;la somma BX è in RISP_MSW
;fine della somma in multipla precisione

          RET                      ;il controllo ritorna al DOS
PROCEDURA ENDP                     ;fine della procedura
CODICE    ENDS                     ;fine del segmento di codice

          END                      ;fine del programma
```

Figura 5.5 Programma che realizza la somma in precisione multipla utilizzando l'indirizzamento diretto

Riportiamo qui di seguito una parte del programma:

```
MOV     AX,NUM1     ;primo numero nel registro AX
ADD     AX,NUM2     ;somma il secondo numero al primo
ADC     BX,00H      ;se il Carry è a 1, viene sommato a BX
ADD     AX,NUM3     ;somma il terzo numero al registro AX
```

```
ADC     BX,00H          ;se il Carry è a 1, viene sommato a BX
ADD     AX,NUM4         ;somma il quarto num. al registro AX
ADC     BX,00H          ;se il Carry è a 1 viene sommato a BX
MOV     RISP_LSW,AX     ;la somma AX è in RISP_LSW
MOV     RISP_MSW,BX     ;la somma BX è in RISP_MSW
```

NUM1 viene sommato a NUM2 utilizzando la semplice sequenza di istruzioni MOV/ADD che abbiamo discusso nel primo esempio di questo capitolo. Se questa somma pone a 1 il bit Carry, la successiva linea di codice (ADC BX,00H) somma il Carry al contenuto del registro BX. La sequenza di istruzioni ADD/ADC viene ripetuta per ogni numero da sommare. Il risultato finale a 32 bit è memorizzato in due registri: BX contiene i bit più significativi, mentre AX contiene i bit meno significativi del numero. Successivamente, il contenuto di questi due registri viene trasferito in due variabili interne al segmento dati – RISP_LSW e RISP_MSW – che, se concatenate, assumono il valore 0002EB1BH. Questa tecnica può essere generalizzata per sommare numeri di qualunque dimensione, ignorando così le reali limitazioni sui valori numerici che possono essere memorizzati effettivamente nei registri di un calcolatore.
Le tecniche di programmazione descritte finora presentano un'altra limitazione: ogni somma o sottrazione richiede una nuova linea di codice (in qualche caso, anche due). Se il numero di somme da eseguire è limitato non si creano problemi, ma se raggiunge valori elevati, il codice diventa ingombrante e poco efficiente.

SOMMA IN PRECISIONE MULTIPLA CON INDIRIZZAMENTO CON REGISTRO INDICE

Il modo più semplice per evitare che ad ogni operazione di somma venga associata una linea di codice è quello di utilizzare:

1. una tabella in memoria che contenga gli addendi;
2. un ciclo di programma che permetta l'esecuzione ripetuta dello stesso codice
3. la modalità di indirizzamento con registro indice per effettuare i trasferimenti all'interno della tabella.

È possibile memorizzare i numeri in tabella sotto un unico nome di variabile, in modo da evitare l'uso di una variabile distinta per referenziare ogni elemento della tabella. Questo concetto diventa comprensibile se esaminiamo il seguente segmento dati:

```
DATI      SEGMENT  PARA 'DATI'
TABELLA   DW       1234H,5678H,9ABCH,0DEF0H,1111H,2222H,3333H
```

```
         DW      4444H,5555H,6666H,7777H,8888H,9999H,0AAAAH
         DW      0BBBBH,0CCCCH,0DDDDH,0EEEEH,0FFFFH
RISP-LSW DW      ?
RISP-MSW DW      ?
DATI     ENDS
```

I numeri vengono referenziati tramite la variabile TABELLA e ognuno di essi è stato definito di tipo word (Define Word). Si noti, in particolare, come la tabella sia stata definita su linee successive. Un programma, per eseguire la somma di alcuni numeri della tabella, deve utilizzare una tecnica che gli permetta prima di referenziare la locazione di memoria TABELLA e poi di selezionare i numeri che gli interessano. La Figura 5.6 mostra il diagramma di flusso dei passi di programma che realizzano l'operazione di somma in multipla precisione utilizzando un ciclo e l'indirizzamento con registro indice. La presenza di un ciclo permette al programma di eseguire – per un numero predefinito di volte – la stessa sequenza di istruzioni, eliminando così la necessità di aggiungere nuove istruzioni per ogni operazione di somma. Il seguente esempio costituisce un programma completo che somma i primi dieci numeri di 16 bit memorizzati in TABELLA:

```
;per macchine 8088/80386
;programma che realizza la somma in multipla precisione
;utilizzando i dati che sono memorizzati in una tabella e un
;indirizzamento con registro indice

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK  ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
TABELLA   DW      1234H,5678H,9ABCH,0DEF0H,1111H,2222H,3333H
          DW      4444H,5555H,6666H,7777H,8888H,9999H,0AAAAH
          DW      0BBBBH,0CCCCH,0DDDDH,0EEEEH,0FFFFH
RISP-LSW  DW      ?
RISP-MSW  DW      ?
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'     ;definisce il segmento di codice
PROCEDURA PROC    FAR               ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS                ;salva DS sullo stack
          SUB     AX,AX             ;azzera AX
          PUSH    AX                ;salva 0 sullo stack
          MOV     AX,DATI           ;indirizzo di DATI in AX
          MOV     DS,AX             ;indirizzo di DATI in DS

;somma in multipla precisione dei primi dieci numeri in tabella
;con indirizzamento con registro indice
          LEA     BX,TABELLA        ;offset di TABELLA in BX
          MOV     SI,00H            ;indice a 0
          MOV     AX,TABELLA[SI]    ;primo numero in AX
          MOV     CX,09H            ;numero di somme nel programma
ANCORA:   ADD     SI,02H            ;punta a una word in TABELLA
          ADD     AX,TABELLA[SI]    ;somma il prossimo numero al risultato parziale
```

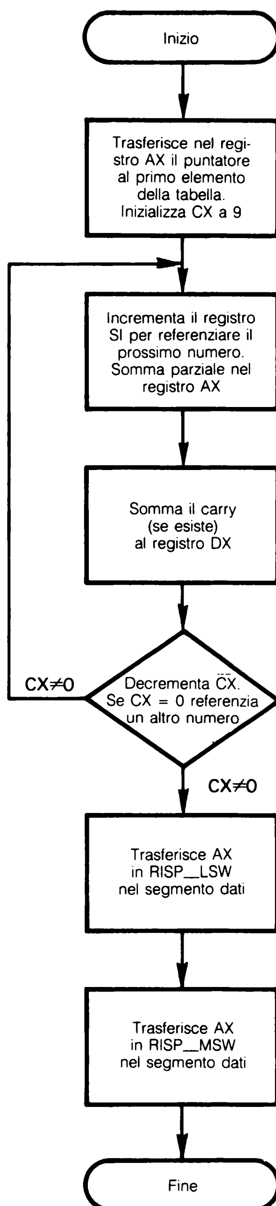


Figura 5.6 Passi del programma che esegue la somma in precisione multipla utilizzando un ciclo e l'indirizzamento con registro indice

```
        ADC     DX,00H          ;se il Carry = 1, viene sommato a DX
        LOOP    ANCORA          ;se CX non è 0, continua
        MOV     RISP_LSW,AX     ;trasferisce AX in RISP_LSW
        MOV     RISP_MSW,DX     ;trasferisce DX in RISP_MSW
;fine della somma in multipla precisione

        RET                     ;il controllo ritorna al DOS
PROCEDURA ENDP                  ;fine della procedura
CODICE  ENDS                    ;fine del segmento di codice

        END                     ;fine del programma
```

Il nucleo del programma è il seguente:

```
         LEA   BX,TABELLA        ;offset di TABELLA in BX
         MOV   SI,00H            ;indice a 0
         MOV   AX,TABELLA[SI]    ;primo numero in AX
         MOV   CX,09H            ;numero di somme nel programma
ANCORA:  ADD   SI,02H            ;punta a una word in TABELLA
         ADD   AX,TABELLA[SI]    ;somma il prossimo numero al risultato
                                 ;parziale
         ADC   DX,00H            ;se il Carry = 1, viene sommato a DX
         LOOP  ANCORA            ;se CX non è 0, continua
```

Il registro indice (SI) contiene lo spiazzamento (riferito a TABELLA) del numero da sommare e viene inizializzato a zero. Il contatore di ciclo assume il valore nove e vengono sommati complessivamente dieci numeri, in quanto il primo numero in TABELLA (1234H) viene trasferito nel registro AX prima che il programma entri in ciclo. Le istruzioni di ciclo sono quelle comprese tra l'etichetta ANCORA e la linea di programma LOOP ANCORA.
Durante la seconda esecuzione del ciclo, il registro indice viene incrementato di 02H e viene di conseguenza referenziato il successivo numero (DW) in TABELLA. Poiché i dati in memoria sono allocati sotto forma di byte, per referenziare la word successiva a quella corrente, il programmatore deve specificarne la dimensione in byte (2). L'istruzione che segue somma il contenuto della locazione di memoria puntata dal registro SI (cioè il numero 5678H) con il contenuto del registro AX e memorizza il risultato dell'operazione sempre nel registro AX. Se la somma produce un overflow, l'istruzione seguente lo accumula nel registro DX.
Il ciclo continua fino a quando il contatore, decrementandosi di una unità ad ogni passaggio nel ciclo, assume il valore 0. In questo caso il programma continua l'esecuzione dall'istruzione seguente a LOOP ANCORA, memorizzando il contenuto finale di DX e AX in due variabili di 16 bit. Il risultato conclusivo (0003 48BDH) può essere visualizzato, richiedendo l'immagine in memoria del segmento dati:

```
-D DS:0000
1CC1:0000   34 12 24 00 BC 0A 54 07-00 00 00 00 00 00 00 00   4.xV......""33DD
1CC1:0010   55 55 66 66 77 77 88 88-99 99 AA AA BB BB CC CC   UUffww..........
```

```
1CC1:0020   DD DD EE EE FF FF BD 48-03 00 00 00 00 00 00 00   .......H........
1CC1:0030   4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20   MYSTACK MYSTACK
1CC1:0040   4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20   MYSTACK MYSTACK
1CC1:0050   4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20   MYSTACK MYSTACK
1CC1:0060   4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20   MYSTACK MYSTACK
1CC1:0070   4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20   MYSTACK MYSTACK
```

SOMMA DI NUMERI DECIMALI CON INDIRIZZAMENTO INDIRETTO CON REGISTRO

L'aritmetica naturale per gli assembler è quella esadecimale, anche se non è escluso che, in certe circostanze, il programmatore preferisca operare con numeri espressi nella notazione decimale. È bene ricordare che la maggior parte degli assembler accettano i dati numerici espressi nella notazione binaria, decimale, esadecimale e ottale, ma questo non significa che le operazioni vengano eseguite indifferentemente in tutte queste basi.
L'assembler converte i numeri nel formato binario quando li alloca in memoria e le operazioni eseguite su basi numeriche diverse da quella esadecimale (binaria) richiedono speciali istruzioni.
La famiglia di microprocessori 8088/80386 supporta due istruzioni (DAA e DAS) che operano in base 10 e il cui significato è stato già discusso nel Capitolo 3. La Figura 5.7 mostra i passi di programma che sono indispensabili per realizzare l'operazione di somma decimale utilizzando l'indirizzamento indiretto con registro.
L'istruzione DAA richiede che gli addendi siano nel formato decimale (ad esempio 3, 4, 8, 11, 12, 30, 89 sono numeri corretti, mentre 1AH, 0AH, 1BH non lo sono), utilizza solo il registro AL, deve essere preceduta sempre da un'istruzione ADD o ADC ed esegue la sistemazione decimale della somma, senza realizzare alcuna conversione effettiva (il calcolatore ragiona ancora in termini esadecimali). Ad esempio, sommando i numeri 5 e 6 si ottiene il risultato esadecimale 0BH; l'istruzione DAA somma a questo risultato 06H e restituisce il valore decimale 11, che viene però memorizzato nel formato esadecimale 11H.
Qui di seguito viene presentato il programma che realizza la somma decimale (si tenga presente che i numeri memorizzati nel segmento dati sono nel formato richiesto dall'operazione di somma decimale).

```
;per macchine 8088/80386
;programma che utilizza l'istruzione DAA per eseguire
;l'operazione di somma con indirizzamento indiretto con registro

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK  ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
INFO      DB      1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
```

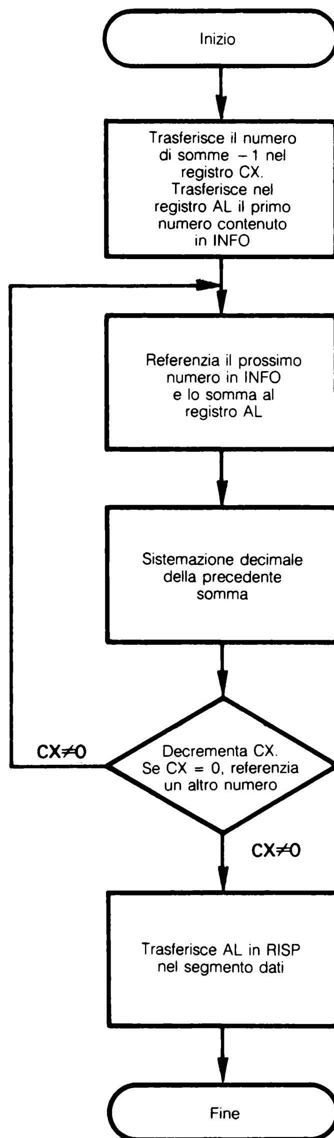


Figura 5.7 Passi di programma per realizzare la somma decimale utilizzando l'indirizzamento indiretto con registro

```
RISP      DB      ?
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'    ;definisce il segmento di codice
PROCEDURA PROC    FAR              ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS               ;salva DS sullo stack
          SUB     AX,AX            ;azzera AX
          PUSH    AX               ;salva 0 sullo stack
          MOV     AX,DATI          ;indirizzo di DATI in AX
          MOV     DS,AX            ;indirizzo di DATI in DS

;somma dei primi undici numeri decimali memorizzati in INFO
          LEA     BX,INFO          ;BX = offset di INFO
          MOV     CX,10            ;numero di somme
          MOV     AL,INFO          ;primo numero da INFO ad AL
ANCORA:   ADD     BX,01H           ;referenzia il prossimo numero
          ADD     AL,INFO[BX]      ;somma il numero al risultato parziale
          DAA                      ;sistemazione decimale della somma
          LOOP    ANCORA           ;se CX non è 0, continua
          MOV     RISP,AL          ;trasferisce la somma in RISP
;fine della somma decimale

          RET                      ;il controllo ritorna al DOS
PROCEDURA ENDP                     ;fine della procedura
CODICE    ENDS                     ;fine del segmento di codice

          END                      ;fine del programma
```

Il nucleo del programma è il seguente:

```
          LEA    BX,INFO        ;BX = offset di INFO
          MOV    CX,10          ;numero di somme
          MOV    AL,INFO        ;primo numero da INFO ad AL
ANCORA:   ADD    BX,01H         ;referenzia il prossimo numero
          ADD    AL,INFO[BX]    ;somma il numero al risultato parziale
          DAA                   ;sistemazione decimale della somma
          LOOP   ANCORA         ;se CX non è 0, continua
```

Il registro CX memorizza il contatore di ciclo. Questa parte di programma differisce da quella dell'esempio precedente, in quanto non utilizza il registro indice per spaziare lungo la tabella, ma il solo incremento di BX è sufficiente per referenziare tutti i numeri della tabella.
Il registro BX viene incrementato solo di una unità in quanto la zona di memoria occupata da ogni numero della tabella INFO è di un byte. La funzione del ciclo è identica a quella già illustrata nell'esempio precedente, ma in questa occasione l'istruzione ADD è seguita dall'istruzione DAA. Conclusa la somma dei primi 11 numeri, il programma memorizza il contenuto del registro AL nella variabile RISP e il controllo ritorna al sistema operativo. Dopo l'esecuzione del programma, è possibile visualizzare la zona di memoria occupata dal segmento dati e verificare così direttamente il valore del risultato della somma. Si tenga presente che i numeri vengono visualizzati nel formato esadecimale:

```
-D DS:0000
1CE5:0000   01 02 03 04 05 06 07 08-09 0A 0B 0C 0D 0E 0F 66   ...............F
1CE5:0010   55 55 66 66 77 77 88 88-99 99 AA AA BB BB CC CC   MYSTACK MYSTACK
1CE5:0020   DD DD EE EE FF FF BD 48-03 00 00 00 00 00 00 00   MYSTACK MYSTACK
1CE5:0030   4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20   MYSTACK MYSTACK
1CE5:0040   4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20   MYSTACK MYSTACK
1CE5:0050   4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20   MYSTACK MYSTACK
1CE5:0060   4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20   MYSTACK MYSTACK
1CE5:0070   4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20   MYSTACK MYSTACK
```

MOLTIPLICAZIONE PER SOMME RIPETUTE

I primi linguaggi assemblatori – come quelli disponibili per la programmazione sui microprocessori Motorola 6800 e 6502 – supportavano solo le operazioni aritmetiche di somma e di sottrazione, ed era quindi compito del programmatore realizzare algoritmi che eseguissero le operazioni di moltiplicazione e di divisione. La famiglia di microprocessori Intel 8088/80386, invece, supporta anche le istruzioni che realizzano – a livello di microprogramma – la moltiplicazione e la divisione.
Prima di esaminare in dettaglio queste istruzioni, consideriamo un programma che esegue una semplice moltiplicazione per somme ripetute.
Sappiamo che il prodotto 5 x 3 può essere calcolato sommando tre volte il valore numerico 5 e questo algoritmo realizza una moltiplicazione per somme ripetute. La maggiore limitazione a questa tecnica programmativa consiste nel tempo impiegato per eseguire le somme. La Figura 5.8 mostra il diagramma di flusso dei passi di programma che sono necessari per moltiplicare due numeri di 8 bit, utilizzando un contatore di ciclo: il moltiplicando viene ripetutamente sommato a se stesso fino a quando il numero di somme eseguite risulta uguale al valore originale del moltiplicatore.
Il seguente programma esegue l'operazione di moltiplicazione nei termini indicati precedentemente:

```
;per macchine 8088/80386
;programma che esegue l'operazione di moltiplicazione per somme
;ripetute

STACK      SEGMENT PARA STACK
           DB      64 DUP ('MYSTACK ')
STACK      ENDS

DATI       SEGMENT PARA 'DATI'
NUM1       DB      2AH
NUM2       DB      78H
PRODOTTO   DW      ?
DATI       ENDS

CODICE     SEGMENT PARA 'CODICE'    ;definisce il segmento di codice
PROCEDURA  PROC    FAR              ;inizio della procedura
           ASSUME  CS:CODICE,DS:DATI,SS:STACK
           PUSH    DS               ;salva DS sullo stack
           SUB     AX,AX            ;azzera AX
           PUSH    AX               ;salva 0 sullo stack
```

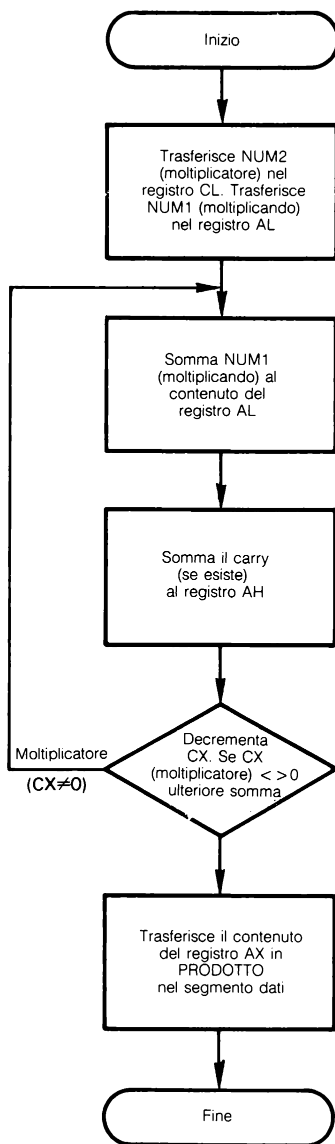


Figura 5.8 Diagramma di flusso per la moltiplicazione di due numeri per somme ripetute

```
          MOV       AX,DATI          ;indirizzo di DATI in AX
          MOV       DS,AX            ;indirizzo di DATI in DS

;istruzioni che realizzano la moltiplicazione per somme ripetute
          SUB       AX,AX            ;azzera il registro AX
          SUB       CX,CX            ;azzera il registro CX
          MOV       CL,NUM2          ;moltiplicatore in CL
          SUB       CL,01H           ;indice corretto
          MOV       AL,NUM1          ;moltiplicando in AL
ANCORA:   ADD       AL,NUM1          ;somma il moltiplicando al risultato parziale
          ADC       AH,00H           ;accumula un totale di 16 bit
          LOOP      ANCORA           ;se CL non è 0, continua
          MOV       PRODOTTO,AX      ;risultato finale in PRODOTTO
;fine della moltiplicazione

          RET                        ;il controllo ritorna al DOS
PROCEDURA ENDP                       ;fine della procedura
CODICE    ENDS                       ;fine del segmento di codice

          END                        ;fine del programma
```

La parte principale del programma è la seguente:

```
         SUB    AX,AX       ;azzera il registro AX
         SUB    CX,CX       ;azzera il registro CX
         MOV    CL,NUM2     ;moltiplicatore in CL
         SUB    CL,01H      ;indice corretto
         MOV    AL,NUM1     ;moltiplicando in AL
ANCORA:  ADD    AL,NUM1     ;somma il moltiplicando al risultato
                            ;parziale
         ADC    AH,00H      ;accumula un totale di 16 bit
         LOOP   ANCORA      ;se CL non è 0, continua
```

I registri AX e CX vengono inizializzati a zero per garantire il corretto ottenimento del risultato. AX è il registro generale in cui viene eseguita la somma, mentre CL contiene il valore del moltiplicatore (cioè il numero di ripetizioni del ciclo) decrementato di una unità, poiché il moltiplicando viene trasferito in AL prima che il programma entri in ciclo.
Quando il ciclo viene eseguito la prima volta, il moltiplicando viene sommato a se stesso. L'istruzione ADC tiene conto del riporto e memorizza in AX un risultato finale di 16 bit (AH contiene i bit più significativi, mentre AL quelli meno significativi). Il ciclo viene ripetuto fino a quando il moltiplicatore, decrementato ogni volta di una unità, assume il valore 0.

MOLTIPLICAZIONE, ELEVAMENTO AL QUADRATO E AL CUBO CON L'ISTRUZIONE DI MOLTIPLICAZIONE

Il prossimo esempio si compone di tre programmi che realizzano le operazioni di moltiplicazione, di elevamento al quadrato e di elevamento al cubo, secondo le modalità indicate in Figura 5.9.

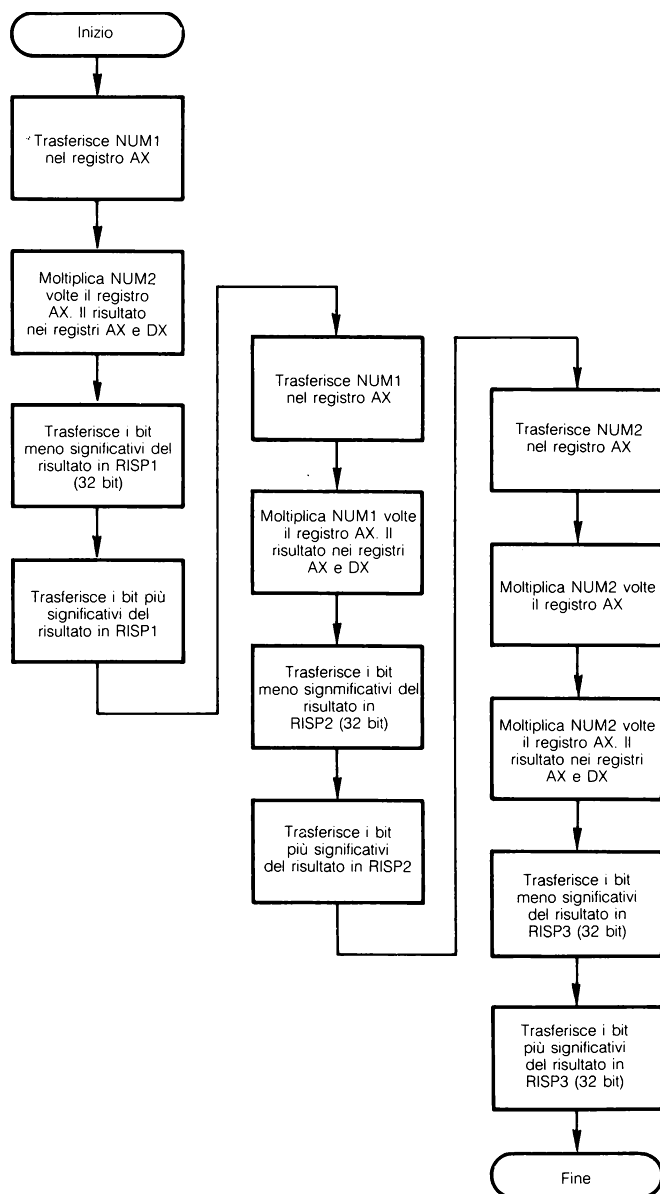


Figura 5.9 Passi di programma per realizzare le operazioni di moltiplicazione, elevamento al quadrato ed elevamento al cubo

Per elevare un numero ad una potenza pari, si può utilizzare un algoritmo efficiente (simile a quello adottato per la moltiplicazione), per cui l'operazione di elevamento a potenza viene realizzata per moltiplicazioni ripetute. Il seguente programma esegue quanto detto:

```
;per macchine 8088/80386
;programma che illustra l'uso del comando di moltiplicazione per
;realizzare le operazioni di moltiplicazione, elevamento al
;quadrato e al cubo. Il massimo numero di cui è possibile
;calcolare la terza potenza, con questa tecnica programmativa, è
;509H.

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
NUM1      DW      1234H
NUM2      DW      00CDH
RISP1     DD      ?
RISP2     DD      ?
RISP3     DD      ?
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'     ;definisce il segmento di codice
PROCEDURA PROC    FAR               ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS                ;salva DS sullo stack
          SUB     AX,AX             ;azzera AX
          PUSH    AX                ;salva 0 sullo stack
          MOV     AX,DATI           ;indirizzo di DATI in AX
          MOV     DS,AX             ;indirizzo di DATI in DS

;moltiplicazione di due numeri con il risultato memorizzato in
;una varibile di tipo doubleword
          SUB     DX,DX             ;azzera il registro di overflow
          MOV     AX,NUM1           ;numero NUM1 in AX
          MUL     NUM2              ;moltiplicazione di NUM1 con NUM2
          MOV     WORD PTR RISP1,AX    ;AX in LSB di RISP1
          MOV     WORD PTR RISP1+2,DX  ;DX in MSB di RISP1
;fine della moltiplicazione

;elevamento a quadrato utilizzando l'istruzione di
;moltiplicazione con il risultato memorizzato in una variabile di
;tipo doubleword
          SUB     DX,DX             ;azzera il registro di overflow
          MOV     AX,NUM1           ;il numero NUM1 in AX
          MUL     NUM1              ;NUM1 moltiplicato per se stesso
          MOV     WORD PTR RISP2,AX    ;AX in LSB di RISP2
          MOV     WORD PTR RISP2+2,DX  ;DX in MSB di RISP2
;fine dell'elevamento al quadrato

;elevamento al cubo utilizzando l'istruzione di moltiplicazione
;con il risultato memorizzato in una variabile doubleword
          SUB     DX,DX             ;azzera il registro di overflow
          MOV     AX,NUM2           ;il numero NUM2 in AX
          MUL     NUM2              ;NUM2 moltiplicato per se stesso
          MUL     NUM2              ;NUM2 moltiplicato per se stesso
          MOV     WORD PTR RISP3,AX    ;AX in LSB di RISP3
```

```
          MOV      WORD PTR RISP3+2,DX ;DX in MSB di RISP3
;fine dell'elevamento al cubo

          RET                          ;il controllo ritorna al DOS
PROCEDURA ENDP                         ;fine della procedura
CODICE    ENDS                         ;fine del segmento di codice

          END                          ;fine del programma
```

Il segmento dati contiene alcune variabili di particolare interesse:

```
DATI      SEGMENT PARA     'DATI'
NUM1      DW      1234H
NUM2      DW      00CDH
RISP1     DD      ?
RISP2     DD      ?
RISP3     DD      ?
DATI      ENDS
```

I due numeri NUM1 e NUM2 occupano una zona di memoria di 16 bit, mentre le tre variabili destinate a contenere i risultati delle operazioni – RISP1, RISP2 e RISP3 – sono di tipo doubleword, per cui occupano 32 bit di memoria. Sappiamo, però, che la famiglia di microprocessori 8088/80286 dispone di registri generali a 16 bit, per cui è indipensabile utilizzare una tecnica programmativa che permetta di memorizzare dati a 32 bit, come viene qui di seguito riportato:

```
SUB       DX,DX                  ;azzera il registro di overflow
MOV       AX,NUM1                ;numero NUM1 in AX
MUL       NUM2                   ;moltiplicazione di NUM1 con NUM2
MOV       WORD PTR RISP1,AX      ;AX in LSB di RISP1
MOV       WORD PTR RISP1+2,DX    ;DX in MSB di RISP1
```

L'istruzione MUL genera un risultato di 32 bit e lo memorizza nei registri DX e AX.
In questo esempio, NUM1 viene trasferito nel registro AX e viene moltiplicato direttamente per NUM2 (non occorre caricare anche NUM2 in un registro). Molte operazioni aritmetiche come la somma, la sottrazione, il decremento e l'incremento possono operare direttamente sulle variabili allocate nel segmento dati, per cui non occorre averle trasferite precedentemente in un registro generale.
Il risultato della moltiplicazione viene memorizzato nei registri DX (000EH) e AX (93A4H). Per salvare il contenuto di DX e AX in una variabile di 32 bit, viene utilizzato l'operatore PTR: l'istruzione MOV PTR RISP1,AX trasferisce il contenuto del registro AX nei primi 16 bit (i meno significativi) della locazione di memoria RISP1, mentre l'istruzione MOV PTR RISP1+2,DX (notare l'incremento di due byte) trasferisce il contenuto del registro DX nei secondi 16 bit (i più significativi) della locazione di memoria RISP1. L'ope-

ratore PTR elimina quindi qualunque ambiguità possa crearsi per l'assembler, in riferimento alla memorizzazione di un risultato di 16 bit in un registro a 32 bit.
La seguente parte di programma evidenzia come eseguire l'operazione di elevamento al quadrato, moltiplicando il numero per se stesso.

```
SUB    DX,DX                   ;azzera il registro di overflow
MOV    AX,NUM1                 ;il numero NUM1 in AX
MUL    NUM1                    ;NUM1 moltiplicato per se stesso
MOV    WORD PTR RISP2,AX       ;AX in LSB di RISP2
MOV    WORD PTR RISP2+2,DX     ;DX in MSB di RISP2
```

In questo caso, il moltiplicando e il moltiplicatore coincidono. Il risultato, memorizzato in DX (014BH) e in AX (5A90H), viene trasferito nella varibile di 32 bit RISP2.
Da ultimo, mostriamo come realizzare l'operazione di elevamento al cubo, moltiplicando semplicemente tre volte il numero per se stesso.

```
SUB    DX,DX                   ;azzera il registro di overflow
MOV    AX,NUM2                 ;il numero NUM2 in AX
MUL    NUM2                    ;NUM2 moltiplicato per se stesso
MUL    NUM2                    ;NUM2 moltiplicato per se stesso
MOV    WORD PTR RISP3,AX       ;AX in LSB di RISP3
MOV    WORD PTR RISP3+2,DX     ;DX in MSB di RISP3
```

Rimane da verificare che il risultato non superi i limiti imposti dalla dimensione della locazione di memoria destinata a contenerlo. Il cubo del numero 0CDH vale 8374D5H e può essere memorizzato in un registro a 32 bit. Fino a quando il numero di cui si intende calcolare la terza potenza non eccede il valore 50AH, non si verificano effetti di overflow.

USO DELL'ISTRUZIONE DI DIVISIONE CON VARIABILI DI TIPO DOUBLEWORD

La Figura 5.10 mostra i passi di programma che sono necessari per realizzare una semplice operazione di divisione, ma non indica la dimensione del dividendo o del divisore. Quando nel Capitolo 3 abbiamo discusso l'istruzione di divisione, abbiamo detto che il dividendo può essere memorizzato in due registri a 16 bit (DX e AX). In questo esempio, il dividendo viene memorizzato in un segmento dati come variabile di tipo doubleword (DD) e il seguente programma ne illustra l'utilizzo:

```
;per macchine 8088/80386
;programma che esegue l'operazione di divisione con il comando
;div
```

```
STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
DIVIDENDO DD      02A8B7654H
DIVISORE  DW      5ABCH
QUOZIENTE DW      ?
RESTO     DW      ?
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'   ;definisce il segmento di codice
PROCEDURA PROC    FAR             ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS              ;salva DS sullo stack
          SUB     AX,AX           ;azzera AX
          PUSH    AX              ;salva 0 sullo stack
          MOV     AX,DATI         ;indirizzo di DATI in AX
          MOV     DS,AX           ;indirizzo di DATI in DS

;istruzioni che realizzano  la divisione tra un dividendo di 32
;bit e un divisore di 16 bit
          MOV     AX,WORD PTR DIVIDENDO   ;LSB di DIVIDENDO in AX
          MOV     DX,WORD PTR DIVIDENDO+2 ;MSB di DIVIDENDO in DX
          DIV     DIVISORE        ;(DX AX) diviso per DIVISORE
          MOV     QUOZIENTE,AX    ;AX in QUOZIENTE
          MOV     RESTO,DX        ;DX in RESTO
;fine dell'esempio di divisione

          RET                     ;il controllo ritorna al DOS
PROCEDURA ENDP                    ;fine della procedura
CODICE    ENDS                    ;fine del segmento di codice

          END                     ;fine del programma
```

Il segmento dati del programma contiene quattro variabili: DIVIDENDO, DIVISORE, QUOZIENTE e RESTO:

```
DATI       SEGMENT   PARA    'DATI'
DIVIDENDO  DD        02A8B7654H
DIVISORE   DW        5ABCH
QUOZIENTE  DW        ?
RESTO      DW        ?
DATI       ENDS
```

La dimensione di ogni variabile viene definita prima che inizi l'esecuzione del programma. DIVIDENDO contiene il valore numerico 02A8B7654H (713782868 in decimale) e DIVISORE contiene il valore numerico 5ABCH (23228 in decimale). Le due variabili QUOZIENTE e RESTO hanno la dimensione di una word (DW) e non sono state inizializzate (simbolo '?').
La seguente parte del programma indica come viene utilizzato il dividendo di 32 bit:

```
MOV   AX,WORD PTR DIVIDENDO     ;LSB di DIVIDENDO in AX
MOV   DX,WORD PTR DIVIDENDO+2   ;MSB di DIVIDENDO in DX
```

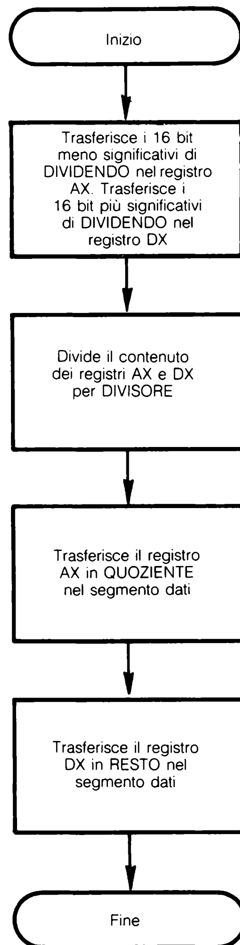


Figura 5.10 Passi del programma che esegue la divisione esadecimale

```
DIV   DIVISORE            ;(DX AX) diviso per DIVISORE
MOV   QUOZIENTE,AX        ;AX in QUOZIENTE
MOV   RESTO,DX            ;DX in RESTO
```

Quando viene utilizzata l'istruzione di divisione, il registro AX deve contenere i bit meno significativi e il registro DX i bit più significativi del dividendo di 32 bit e, per soddisfare questa necessità, il programma utilizza

l'operando WORD PTR. (Specificando WORD PTR DIVIDENDO e WORD PTR DIVIDENDO+2 è possibile caricare rispettivamente i registri AX e DX, poiché il microprocessore riconosce a livello hardware il byte quale unità di informazione trasferibile).
In questo programma, inoltre, il divisore viene direttamente ricavato dal segmento dati, il quoziente – 7809H (30729 in decimale) – viene memorizzato in AX e il resto – 25B8H (9656 in decimale) – viene memorizzato in DX.

UN ALGORITMO PER L'ESTRAZIONE DI RADICE

Il primo algoritmo che abbiamo presentato in questo capitolo realizzava l'operazione di moltiplicazione per somme ripetute, ma aveva solo una funzione didattica in quanto la famiglia di microprocessori 8088/80386 supporta l'istruzione di moltiplicazione. Il seguente esempio, invece, rappresenta un algoritmo che è indispensabile per realizzare l'estrazione di radice di un numero, in quanto questa istruzione non è supportata dai microprocessori Intel. La Figura 5.11 mostra i passi di programma che sono necessari per realizzare l'operazione di estrazione di radice di un numero esadecimale, utilizzando un algoritmo molto semplice che prevede di approssimare il valore della radice quadrata con il numero di valori dispari crescenti che è possibile sottrarre al numero di partenza, fino a quando il risultato corrente della sottrazione non assume il valore 0. Ad esempio, per calcolare la radice quadrata del numero decimale 82 procediamo come segue:

```
82 -  1 = 81
81 -  3 = 78
78 -  5 = 73
73 -  7 = 66
66 -  9 = 57
57 - 11 = 46
46 - 13 = 33
33 - 15 = 18
18 - 17 =  1 ← nove sottrazioni, prima che il risultato sia < 0
 1 - 19 = -18
```

In questo modo, il valore approssimato della radice quadrata di 82 vale 9. Lo stesso algoritmo opera sui numeri che sono espressi nel formato esadecimale. Il seguente programma indica come viene realizzata l'operazione di estrazione di radice nel linguaggio assemblatore:

```
;per macchine 8088/80386
;programma che illustra una tecnica programmativa per eseguire
;l'estrazione di radice di un numero esadecimale

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
```

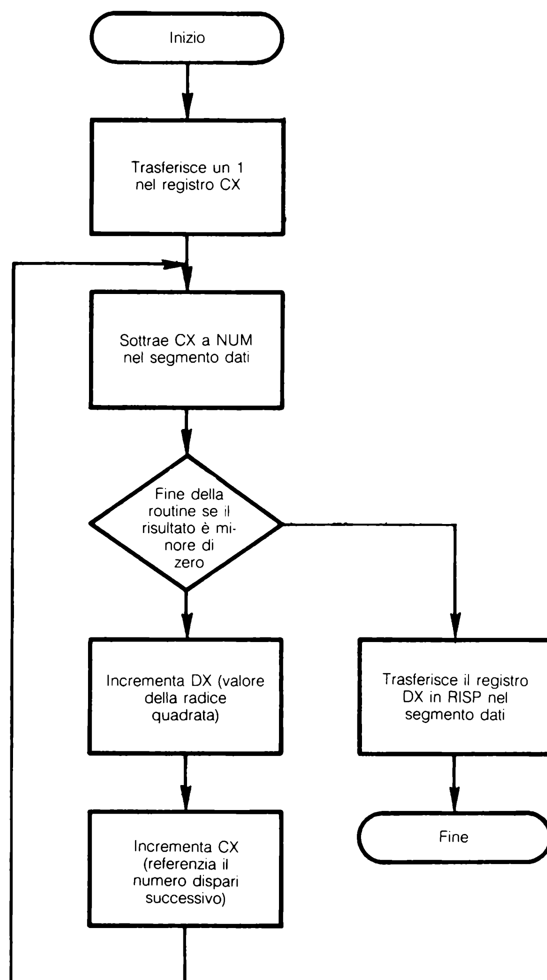


Figura 5.11 Passi di programma per realizzare l'operazione di estrazione di radice di un numero esadecimale

```
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
NUM       DW       1CE4H
RISP      DW       ?
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'    ;definisce il segmento di codice
```

```
PROCEDURA PROC    FAR                 ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS                  ;salva DS sullo stack
          SUB     AX,AX               ;azzera AX
          PUSH    AX                  ;salva 0 sullo stack
          MOV     AX,DATI             ;indirizzo di DATI in AX
          MOV     DS,AX               ;indirizzo di DATI in DS

;codice che esegue l'estrazione di radice
          MOV     DX,0H               ;azzera RISP
          MOV     CX,01H              ;numero da sottrarre
ANCORA:   SUB     NUM,CX              ;sottrazione di un numero dispari
          JL      OUT                 ;esce
          INC     DX                  ;incrementa il valore della radice quadrata
          ADD     CX,02H              ;in CX c'è il nuovo numero da sottrarre
          JMP     ANCORA              ;ripete
OUT:      MOV     RISP,DX             ;il risultato è in RISP
;fine dell'esempio di estrazione di radice di un numero esadecimale

          RET                         ;il controllo ritorna al DOS
PROCEDURA ENDP                        ;fine della procedura
CODICE    ENDS                        ;fine del segmento di codice

          END                         ;fine del programma
```

Il codice che realizza l'algoritmo di estrazione di radice contiene un ciclo:

```
         MOV   DX,0H     ;azzera RISP
         MOV   CX,01H    ;numero da sottrarre
ANCORA:  SUB   NUM,CX    ;sottrazione di un numero dispari
         JL    OUT       ;esce
         INC   DX        ;incrementa il valore della radice quadrata
         ADD   CX,02H    ;in CX c'è il nuovo numero da sottrarre
         JMP   ANCORA    ;ripete
OUT:     MOV   RISP,DX   ;il risultato è in RISP
```

In questo esempio, DX viene inizializzato a zero, in quanto è destinato a contenere il risultato (cioè il valore della radice quadrata). CX memorizza il numero dispari, di valore via via crescente, che deve essere sottratto al numero di partenza e contiene inizialmente il valore 1. Quando il programma entra nel ciclo, infatti, il numero NUM viene diminuito del valore contenuto in CX e, se il risultato di questa sottrazione è minore di zero, il programma termina; in caso contrario, CX viene incrementato al numero dispari successivo e il ciclo continua. Il risultato finale viene memorizzato nella variabile RISP.

5.2 Operazioni logiche

In alcune circostanze, la programmazione in linguaggio assemblatore presuppone un utilizzo delle funzioni logiche (OR, AND e XOR) che si differenzia da quello consueto. Ad esempio, la funzione XOR (OR esclusivo) viene

frequentemente utilizzata nei programmi di grafica per cancellare i punti precedentemente visualizzati sullo schermo, mentre la funzione AND permette di isolare alcuni bit in un dato. Entrambe queste applicazioni verranno discusse in dettaglio nei prossimi paragrafi.

SIMULAZIONE DI OPERAZIONI E DI PORTE LOGICHE IN LINGUAGGIO ASSEMBLATORE

Un'altra applicazione delle funzioni logiche è quella di simulare semplici circuiti logici. La Figura 5.12 mostra un circuito che si compone di quattro porte logiche. I quattro ingressi – IN1, IN2, IN3 e IN4 – rappresentano le connessioni elettriche che costituiscono gli ingressi al circuito, mentre la variabile RISP costituisce la funzione di uscita che viene prodotta dal circuito. Si può analizzare il funzionamento del circuito costruendo la tavola della verità. Il nostro esempio contiene la dichiarazione di un segmento dati in cui sono stati definiti i quattro ingressi come variabili binarie. Il circuito – cioè il programma – analizza i singoli bit degli ingressi e genera la funzione di uscita, procedendo come segue:

```
IN1     DB      10111010B
IN2     DB      11111100B
IN3     DB      00010100B
IN4     DB      00010011B
RISP    DB      ?
```

Innanzitutto, i bit di IN1 vengono complementati con un invertitore logico, ottenendo così il valore 01000101B che, insieme a IN2, costituisce l'ingresso della porta OR. L'operazione di somma logica produce come risultato:

```
01000101B
11111100B
---------
11111101B   (uscita dalla porta OR)
```

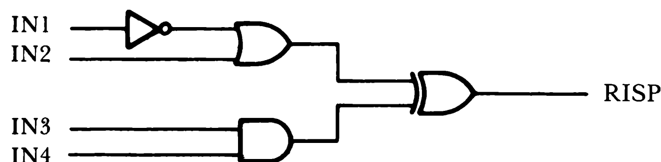


Figura 5.12 Semplice circuito logico, il cui funzionamento può essere simulato software

Successivamente, IN3 e IN4 confluiscono direttamente nella porta AND, che genera in uscita:

```
 00010100B
 00010011B
 ---------
 00010000B  (uscita dalla porta AND)
```

Infine, le uscite dalla porta OR e dalla porta AND costituiscono gli ingressi della porta XOR, per cui il risultato finale è il seguente:

```
 11111101B  (uscita dalla porta OR)
 00010000B  (uscita dalla porta AND)
 ---------
 11101101B  (uscita dalla porta XOR)
```

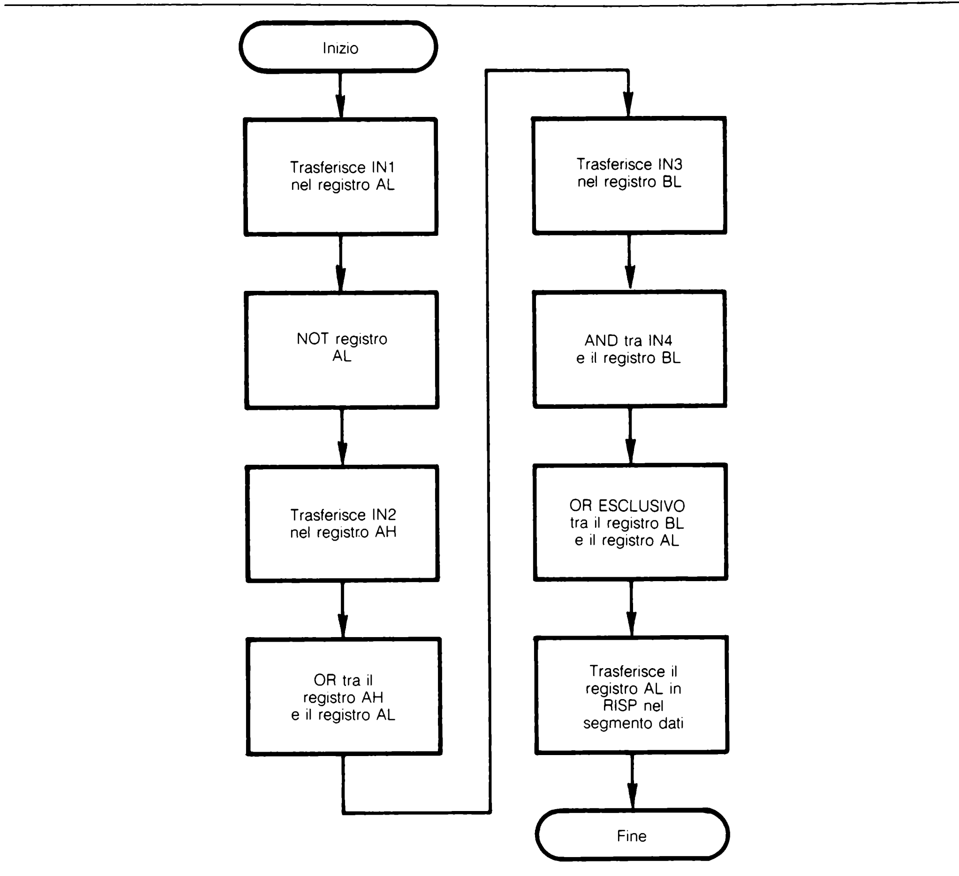


Figura 5.13 Passi di programma necessari per simulare il circuito di Figura 5.12

La Figura 5.13 mostra i passi di programma che sono necessari per simulare a livello software la funzione realizzata da un circuito logico.
Per rendere il programma di facile comprensione, abbiamo definito gli ingressi come variabili binarie di tipo byte (DB), ma, se utilizziamo il programma DEBUG della IBM per visualizzare la zona di memoria occupata dal segmento dati, si nota che le variabili in esso contenute appaiono nel formato esadecimale.
Il seguente programma simula il funzionamento del circuito di Figura 5.12.

```
;per macchine 8088/80386
;programma che simula il funzionamento di un circuito logico
;costituito da porte AND, OR, XOR e NOT

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
IN1       DB   10111010B
IN2       DB   11111100B
IN3       DB   00010100B
IN4       DB   00010011B
RISP      DB   ?
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'     ;definisce il segmento di codice
PROCEDURA PROC    FAR               ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS                ;salva DS sullo stack
          SUB     AX,AX             ;azzera AX
          PUSH    AX                ;salva 0 sullo stack
          MOV     AX,DATI           ;indirizzo di DATI in AX
          MOV     DS,AX             ;indirizzo di DATI in DS

;esempio di sintesi di porte logiche
          MOV     AL,IN1            ;carica il primo ingresso in AL
          NOT     AL                ;complementa i bit di AL
          MOV     AH,IN2            ;carica il secondo ingresso in AH
          OR      AL,AH             ;somma logica di NUM1 negato e NUM2
          MOV     BL,IN3            ;carica il terzo ingresso in BL
          AND     BL,IN4            ;prodotto logico di IN3 e IN4
          XOR     AL,BL             ;OR esclusivo delle uscite delle porte OR e AND
          MOV     RISP,AL           ;risultato in RISP
;fine dell'esempio di sintesi di porte logiche

          RET                       ;il controllo ritorna al DOS
PROCEDURA ENDP                      ;fine della procedura
CODICE    ENDS                      ;fine del segmento di codice

          END                       ;fine del programma
```

La parte di programma qui di seguito riportata esegue alcune operazioni molto semplici: legge gli ingressi da registri a 8 bit, esegue le operazioni logiche richieste e memorizza il risultato finale nella variabile RISP con il formato esadecimale:

```
MOV    AL,IN1     ;carica il primo ingresso in AL
NOT    AL         ;complementa i bit di AL
```

```
MOV   AH,IN2     ;carica il secondo ingresso in AH
OR    AL,AH      ;somma logica di NUM1 negato e NUM2
MOV   BL,IN3     ;carica il terzo ingresso in BL
AND   BL,IN4     ;prodotto logico di IN3 e IN4
XOR   AL,BL      ;OR esclusivo delle uscite delle porte OR e AND
MOV   RISP,AL    ;risultato in RISP
```

5.3 Tabelle di ricerca (lookup)

Una tabella di ricerca (lookup) permette al programmatore di ricavare – mediante una ricerca tabellare – un valore o un insieme di dati, velocizzando così la scrittura e l'esecuzione di programmi grafici e di programmi di ricerca di elementi in un insieme. Ad esempio, è possibile memorizzare in un segmento dati un intero dizionario di parole in forma tabellare e scandirlo successivamente confrontando le parole che lo compongono con la parola da cercare. (Il Capitolo 8 contiene un semplice programma che esegue questa ricerca utilizzando una tabella di lookup).
In questo capitolo, vengono presentati due esempi che evidenziano l'utilità delle tabelle di consultazione.

TABELLA DI LOOKUP PER LOGARITMI

Pensare, e poi scrivere, gli algoritmi che realizzano particolari funzioni matematiche risulta molto difficile e l'uso di un coprocessore matematico rappresenta la migliore soluzione per risolvere questo problema.
Occupiamoci ora di illustrare la funzionalità delle tabelle di lookup nel calcolo dei risultati di alcune operazioni aritmetiche.
La Figura 5.14 mostra i passi di programma che sono richiesti per determinare il logaritmo di un numero, dopo aver calcolato e memorizzato in forma tabellare i valori della funzione logaritmo per un predefinito insieme di numeri. Il seguente segmento dati contiene una TABELLA in cui sono memorizzati i valori del logaritmo dei numeri interi decimali compresi tra 1 e 10.

```
TABELLA    DW    0,3010,4771,6021,6990,7782,8451,9031,9542,10000
VALORE     DB    7
RISPOSTA   DW    ?
```

Ogni numero in tabella possiede implicitamente la virgola decimale. Ad esempio, il numero 4771 rappresenta il valore 0.4771 del logaritmo. Inoltre, si tenga presente che questi numeri sono definiti nel formato decimale e occupano

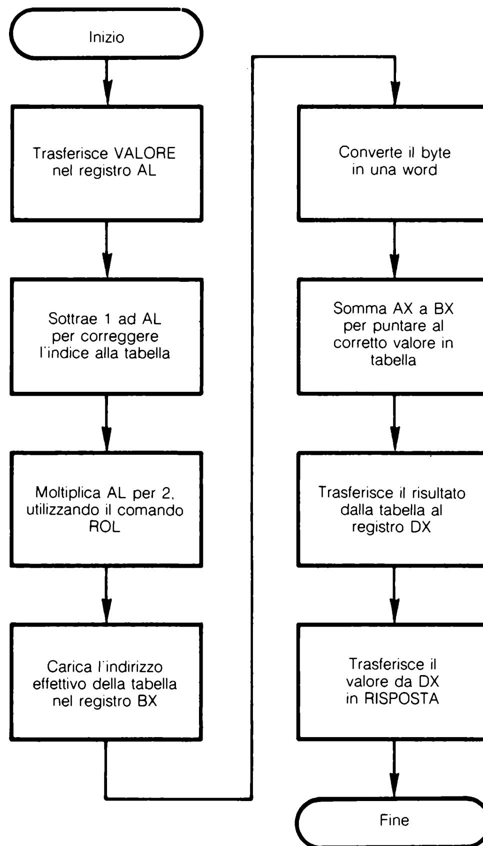


Figura 5.14 Passi di programma necessari per ricavare il logaritmo di un numero

una word di memoria (DW). Il programma che calcola il logaritmo di un numero decimale intero compreso tra 1 e 10 è dunque il seguente:

```
;per macchine 8088/80386
;programma che illustra l'uso di una tabella di lookup nel
;calcolo del logaritmo di un numero decimale intero compreso tra
;1 e 10

STACK      SEGMENT PARA STACK
           DB      64 DUP ('MYSTACK ')
STACK      ENDS

DATI       SEGMENT PARA 'DATI'
TABELLA    DW    0,3010,4771,6021,6990,7782,8451,9031,9542,10000
```

```
VALORE    DB    7
RISPOSTA  DW    ?
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'      ;definisce il segmento di codice
PROCEDURA PROC    FAR                ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS                 ;salva DS sullo stack
          SUB     AX,AX              ;azzera AX
          PUSH    AX                 ;salva 0 sullo stack
          MOV     AX,DATI            ;indirizzo di DATI in AX
          MOV     DS,AX              ;indirizzo di DATI in DS

;viene utilizzato un valore predefinito come indice alla tabella
;di lookup
          MOV     AL,VALORE          ;indice per la ricerca
          SUB     AL,1               ;indice corretto, no log di 0
          ROL     AL,1               ;indice per 2, per dati di tipo word
          LEA     BX,TABELLA         ;offset di TABELLA in BX
          CBW                        ;converte AL da byte a word
          ADD     BX,AX              ;somma offset + indice
          MOV     DX,TABELLA[BX]     ;il dato in tabella viene estratto
          MOV     RISPOSTA,DX        ;risultato in RISPOSTA
;fine dell'esempio di ricerca

          RET                        ;il controllo ritorna al DOS
PROCEDURA ENDP                       ;fine della procedura
CODICE    ENDS                       ;fine del segmento di codice

          END                        ;fine del programma
```

Per semplificare il programma, abbiamo memorizzato nella variabile VALORE l'indice all'elemento della tabella che contiene il logaritmo che interessa, mentre il risultato viene memorizzato nella variabile di tipo word (DW) RISPOSTA. Esaminiamo dettagliatamente il nucleo del programma:

```
MOV   AL,VALORE          ;indice per la ricerca
SUB   AL,1               ;indice corretto, no log di 0
ROL   AL,1               ;indice per 2, per dati di tipo word
LEA   BX,TABELLA         ;offset di TABELLA in BX
CBW                      ;converte AL da byte a word
ADD   BX,AX              ;somma offset + indice
MOV   DX,TABELLA[BX]     ;il dato in tabella viene estratto
MOV   RISPOSTA,DX        ;risultato in RISPOSTA
```

Nel registro AL viene memorizzato, dopo alcune operazioni di inizializzazione, l'indice alla tabella di consultazione. Poiché non è possibile calcolare il logaritmo di zero, il contenuto di AL viene decrementato di una unità. Inoltre, avendo definito gli elementi della tabella di tipo word, è necessario moltiplicare l'indice per due, utilizzando, invece dell'istruzione MUL, il semplice comando ROL (rotazione di bit a sinistra), che esegue più velocemente la stessa funzione. L'indice alla tabella occupa un byte di memoria e, poiché deve essere trasferito nel registro BX a 16 bit, viene convertito (istruzione CBW)

nel formato word. Sommando il contenuto di AX (indice all'elemento in tabella) con il contenuto di BX (offset di inizio tabella) si ottiene infine la posizione in cui è allocato il valore del logaritmo richiesto (8541), che viene trasferito nel registro DX e successivamente nella variabile RISPOSTA.
I vantaggi di utilizzare una tabella di lookup sono i seguenti:

1. un codice relativamente semplice
2. la possibilità di ottenere risultati di elevata precisione nel formato decimale o esadecimale (o in un altro formato).

Gli svantaggi, invece, possono non apparire così immediati:

1. i risultati devono essere precedentemente memorizzati nel segmento dati, e questo limita la flessibilità del programma (ad esempio, nel nostro caso, se l'utente desidera calcolare il logaritmo di 23, è indispensabile modificare il programma);
2. l'accuratezza e la precisione del risultato viene stabilita dal programmatore e non da una funzione matematica. Possono, inoltre, verificarsi errori di memorizzazione, soprattutto se le tabelle contengono molti dati.

CONVERSIONI DI CODICE MEDIANTE TABELLE DI LOOKUP

Nella programmazione in linguaggio assemblatore, è frequente la necessità di convertire una codifica (ad esempio, ASCII, Gray, Excess-3, BCD) – o una base numerica – in un'altra. La conversione tra codifiche pesate, o tra basi numeriche diverse, risulta abbastanza semplice, al contrario della conversione tra codifiche non pesate. Il prossimo esempio, l'ultimo in questo capitolo che illustri la funzione delle tabelle di lookup, è un programma che esegue la conversione di un numero esadecimale nella codifica Gray equivalente.
La codifica Gray riveste un'importanza notevole quando è richiesta una conversione rapida e riduce la possibilità che si verifichino errori di conversione nel formato binario, poiché – nella codifica Gray – un numero si differenzia dal successivo per il valore di un solo bit, come viene qui si seguito illustrato:

```
GRAY      DB   0010B,0110B,0111B,0101B,0100B,1100B,1101B
          DB   1111B,1110B,1010B
VALORE    DB   8H
RISPOSTA  DB   ?
```

Abbiamo definito in un segmento dati, con il formato binario, le codifiche Gray equivalenti alle cifre decimali o esadecimali comprese tra 0 e 9.
La Figura 5.15 mostra i passi di programma che sono indispensabili per rea-

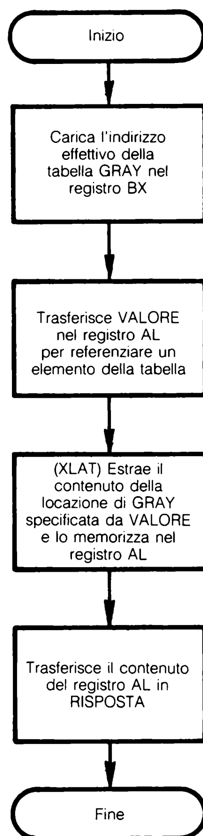


Figura 5.15 Passi di programma necessari per convertire un numero esadecimale nella codifica Gray equivalente

lizzare la conversione di un numero esadecimale nella codifica Gray equivalente. Abbiamo preferito utilizzare l'istruzione XLAT (invece di adottare una soluzione simile a quella presentata nel precedente esempio), che permette di referenziare (tramite il registro AL, per indici di un byte) un elemento di una tabella e memorizzarne automaticamente il valore in un registro (AL). Con semplici modifiche, il seguente programma può eseguire la conversione di numeri esadecimali di tipo word nella codifica Gray equivalente.

```
;per macchine 8088/80386
;programma che illustra l'uso di una tabella di lookup e
```

```
;dell'istruzione XLAT per ricavare la codifica Gray equivalente
;di un numero esadecimale

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
GRAY      DB   0010B,0110B,0111B,0101B,0100B,1100B,1101B
          DB   1111B,1110B,1010B
VALORE    DB   8H
RISPOSTA  DB   ?
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'     ;definisce il segmento di codice
PROCEDURA PROC    FAR               ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS                ;salva DS sullo stack
          SUB     AX,AX             ;azzera AX
          PUSH    AX                ;salva 0 sullo stack
          MOV     AX,DATI           ;indirizzo di DATI in AX
          MOV     DS,AX             ;indirizzo di DATI in DS

;esempio di utilizzo dell'istruzione XLAT per accedere ad un
;elemento di tipo byte della tabella di lookup
          LEA     BX,GRAY           ;offset di inizio della tabella in BX
          MOV     AL,VALORE         ;numero da convertire (indice tabella) in AL
          XLAT    GRAY              ;codifica Gray equivalente in AL
          MOV     RISPOSTA,AL       ;risultato in RISPOSTA
;fine dell'esempio di utilizzo dell'istruzione XLAT

          RET                       ;il controllo ritorna al DOS
PROCEDURA ENDP                      ;fine della procedura
CODICE    ENDS                      ;fine del segmento di codice

          END                       ;fine del programma
```

Si tenga presente che il nucleo del programma si compone solo di quattro istruzioni (qui di seguito riportate), di cui le prime due referenziano in tabella la codifica Gray equivalente del numero esadecimale.

```
LEA    BX,GRAY        ;offset di inizio della tabella in BX
MOV    AL,VALORE      ;numero da convertire (indice tabella) in AL
XLAT   GRAY           ;codifica Gray equivalente in AL
MOV    RISPOSTA,AL    ;risultato in RISPOSTA
```

In questo esempio, la variabile VALORE è stata inizializzata a 08H e costituisce l'indice alla tabella di lookup. Al termine del programma, la variabile RISPOSTA contiene la codifica Gray equivalente – cioè 1110B – che viene però memorizzata nel segmento dati con il formato esadecimale (0EH).

CONVERSIONE DA NUMERI ASCII A NUMERI ESADECIMALI

Si può realizzare la conversione da numeri ASCII a numeri esadecimali utilizzando una tabella di lookup. In questo paragrafo, però, la conversione viene eseguita con l'ausilio di semplici operazioni aritmetiche. L'informazione che viene inviata da tastiera su video o su stampante è codificata in ASCII e non in esadecimale, per cui, se il programmatore preme un tasto numerico (ad esempio 5) il calcolatore ne interpreta la codifica ASCII (cioè 35H). La Tabella 5.1 mostra la corrispondenza esistente tra codifica esadecimale e codifica ASCII.
Se consideriamo i codici ASCII compresi tra 30H e 39H, è sufficiente sottrarre loro 30H per ricavare la corrispondente codifica esadecimale, mentre per i codici compresi tra 41H e 46H, occorre sottrarre 37H (si ricordi di utilizzare l'aritmetica esadecimale per eseguire le sottrazioni). La Figura 5.16 mostra i passi di programma che sono indispensabili per realizzare la conversione dalla codifica ASCII alla codifica esadecimale.
Il seguente esempio illustra come eseguire questa conversione solo sui valori numerici compresi tra 0 e 15 (nonostante la codifica ASCII contenga un campo di valori nettamente superiore), ma non controlla che il codice ASCII inserito dal programmatore corrisponda effettivamente a uno di questi numeri.

```
;per macchine 8088/80386
;programma che esegue la conversione da numeri ASCII a numeri
;esadecimali
```

Tabella 5.1 Corrispondenza tra le codifiche ASCII ed esadecimale

ASCII	**Esadecimale**
30H	0H
31H	1H
32H	2H
33H	3H
34H	4H
35H	5H
36H	6H
37H	7H
38H	8H
39H	9H
41H	AH
42H	BH
43H	CH
44H	DH
45H	EH
46H	FH

```
STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
ASCII     DB   42H            ;carattere 'B' nella codifica ASCII
RISPOSTA  DB   ?
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'     ;definisce il segmento di codice
PROCEDURA PROC    FAR               ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS                ;salva DS sullo stack
          SUB     AX,AX             ;azzera AX
          PUSH    AX                ;salva 0 sullo stack
          MOV     AX,DATI           ;indirizzo di DATI in AX
          MOV     DS,AX             ;indirizzo di DATI in DS

;codice di conversione da ASCII a esadecimale / non viene
;effettuato nessun controllo di errore sul dato in ingresso, ma
;si suppone che questo codifichi uno dei numeri esadecimali
;0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
          MOV     AL,ASCII          ;numero ASCII da convertire in AL
          SUB     AL,30H            ;conversione
          CMP     AL,9              ;il numero e compreso tra 0 e 9?
          JG      LETTERA           ;il numero è maggiore di 9
          JMP     FINE              ;esce
LETTERA:  SUB     AL,07H            ;conversione per numeri maggiori di 9
FINE:     MOV     RISPOSTA,AL       ;risultato in RISPOSTA
;fine dell'esempio di conversione

          RET                       ;il controllo ritorna al DOS
PROCEDURA ENDP                      ;fine della procedura
CODICE    ENDS                      ;fine del segmento di codice

          END                       ;fine del programma
```

Nella parte di programma che segue, si noti come venga sottratto immediatamente al codice ASCII il valore 30H, in quanto si presuppone che l'ingresso sia compreso tra i valori ASCII 30H e 46H. Se il risultato della sottrazione fornisce un valore compreso tra 0H e 9H, il programma termina (il codice ASCII corrisponde, in questo caso, ad un numero non maggiore di 9); in caso contrario, il carattere ASCII identifica – nella codifica esadecimale – una lettera, per cui è indispensabile diminuire di 07H il valore del risultato che è stato ottenuto dalla precedente sottrazione (in totale viene sottratto così 37H al carattere ASCII). In questo modo, si ricava la corretta codifica esadecimale di un numero compreso tra 10 e 15.

```
          MOV   AL,ASCII      ;numero ASCII da convertire in AL
          SUB   AL,30H        ;conversione
          CMP   AL,9          ;il numero è compreso tra 0 e 9?
          JG    LETTERA       ;il numero è maggiore di 9
          JMP   FINE          ;esce
LETTERA:  SUB   AL,07H        ;conversione per numeri maggiori di 9
FINE:     MOV   RISPOSTA,AL   ;risultato in RISPOSTA
```

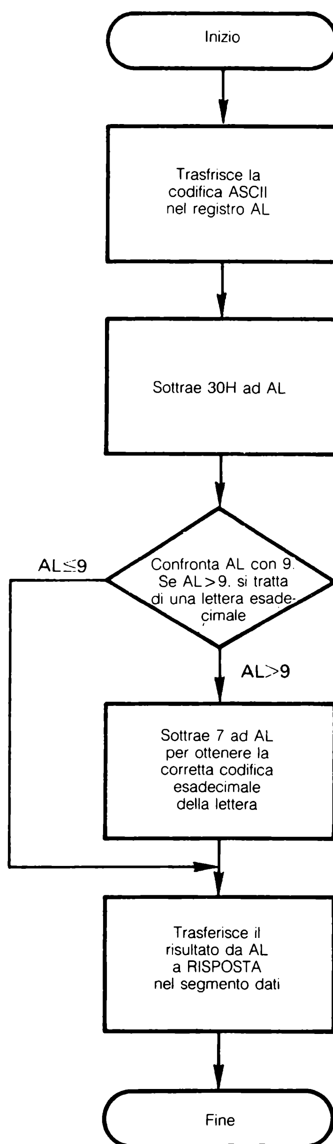


Figura 5.16 Passi di programma per realizzare la conversione dalla codifica ASCII alla codifica esadecimale

5.4 Semplice aritmetica a 32 bit su microprocessore 80386

In un precedente esempio (Figura 5.5) abbiamo ottenuto una precisione numerica a 32 bit utilizzando l'aritmetica in multipla precisione. La famiglia 8088/80286 dispone di registri a 16 bit che possono contenere numeri interi di valore non superiore a 0FFFFH (65 535 in decimale). L'80386, invece, è in grado di memorizzare nei suoi registri generali (EAX, EBX, ECX e EDX) numeri interi fino a 32 bit, cioè di valore non superiore a 0FFFFFFFFH (4 294 967 295 in decimale). Vengono così migliorate sensibilmente le prestazioni rispetto ai precedenti microprocessori.
I prossimi due esempi illustrano come l'80386 esegua le operazioni aritmetiche su numeri molto grandi, senza ricorrere all'aritmetica in multipla precisione.
La Figura 5.17 mostra i passi di programma che sono indispensabili per sommare alcuni numeri aventi una dimensione di 32 bit. Confrontando la Figura 5.17 con la Figura 5.4, si può facilmente constatare la maggiore semplicità programmativa di questo esempio rispetto alle soluzioni precedenti. Il seguente programma dimostra come realizzare la somma di numeri a 32 bit.

```
;per macchine 8088/80386
;programma che esegue la somma a 32 bit sul microprocessore 80386

STACK     SEGMENT PARA STACK
          DB     64 DUP ('MYSTACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
NUMERI    DD     12345678H,9ABCDEF0H,23H,10000000H,0CADH
RISPOSTA  DD     ?
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'    ;definisce il segmento di codice
PROCEDURA PROC   FAR               ;inizio della procedura
          ASSUME CS:CODICE,DS:DATI,SS:STACK
          PUSH   DS                ;salva DS sullo stack
          SUB    AX,AX             ;azzera AX
          PUSH   AX                ;salva 0 sullo stack
          MOV    AX,DATI           ;indirizzo di DATI in AX
          MOV    DS,AX             ;indirizzo di DATI in DS

;codice che esegue la somma a 32 bit dei primi tre numeri
;memorizzati nella variabile NUMERI
          LEA    BX,NUMERI         ;offset di NUMERI in BX
          MOV    EAX,[BX]          ;carica il primo numero di 32 bit in EAX
          ADD    EAX,[BX]+4        ;somma il prossimo numero di 32 bit a EAX
          ADD    EAX,[BX]+8        ;somma il terzo numero di 32 bit a EAX
          MOV    RISPOSTA,EAX      ;risultato in RISPOSTA
;fine dell'esempio di somma a 32 bit

          RET                      ;il controllo ritorna al DOS
PROCEDURA ENDP                     ;fine della procedura
```

```
CODICE   ENDS                  ;fine del segmento di codice

         END                   ;fine del programma
```

Questo programma contiene diversi concetti che probabilmente risultano sconosciuti a chi ha programmato solo con la famiglia di microprocessori Intel 8088/80286.
Esaminiamo il segmento dati:

```
DATI       SEGMENT    PARA     'DATI'
NUMERI     DD         12345678H,9ABCDEF0H,23H,10000000H,0CADH
RISPOSTA   DD         ?
DATI       ENDS
```

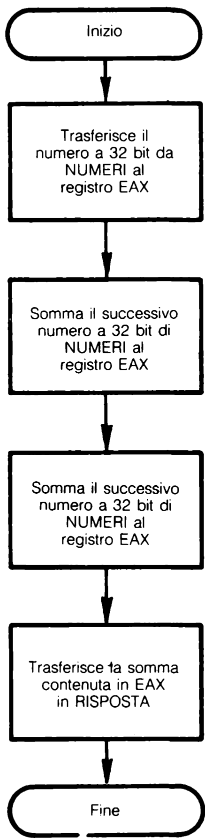


Figura 5.17 Passi del programma che realizza la somma sull'80386

I numeri da sommare (referenziabili attraverso la variabile NUMERI) hanno una dimensione di 32 bit ciascuno, in quanto sono stati definiti di tipo doubleword (DD). Nei precedenti esempi, era impossibile caricare direttamente nei registri di CPU numeri di questa dimensione, mentre ora è lecito codificare le seguenti istruzioni:

```
MOV   EAX,[BX]       ;carica il primo numero di 32 bit in EAX
ADD   EAX,[BX]+4     ;somma il prossimo numero di 32 bit a EAX
ADD   EAX,[BX]+8     ;somma il terzo numero di 32 bit a EAX
MOV   RISPOSTA,EAX   ;risultato in RISPOSTA
```

Il primo numero (12345678H) viene caricato direttamente nel registro EAX con la semplice istruzione MOV. Il secondo numero (9ABCDEF0H) viene referenziato utilizzando un offset di 4 byte rispetto all'indirizzo di base della tabella (posizione 0). Così facendo, infatti, viene puntato il quinto byte del segmento dati, cioè il primo byte in cui è allocato il numero che si intende sommare al contenuto di EAX; con lo stesso procedimento viene sommato al contenuto corrente del registro EAX anche il terzo numero (23H). Infine, il contenuto del registro EAX viene trasferito, con la semplice istruzione MOV, nella variabile RISPOSTA (che assume quindi il valore 0BCF14238H, cioè 3 169 927 736 in decimale).
La Figura 5.18 mostra i passi di programma che sono necessari per moltiplicare due numeri di 32 bit; il risultato ottenuto (un numero esadecimale di valore compreso tra 0H e 0FFFFFFFFFFFFFFFFH, cioè tra 0 e circa 1.8446744073709553E19 in decimale) può essere – nell'80386 – memorizzato direttamente nei registri EDX:EAX, oppure, utilizzando l'operatore PTR, in una locazione di memoria che è stata definita precedentemente di tipo quadword (DQ). Il seguente programma esegue l'operazione di moltiplicazione tra due numeri interi di grandi dimensioni.

```
;per macchine 8088/80386
;programma che illustra una semplice moltiplicazione tra due
;numeri di 32 bit. Il risultato è di 64 bit

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
NUM1      DD      12345678H
NUM2      DD      9ABCDEF0H
RISPOSTA  DQ      ?
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'     ;definisce il segmento di codice
PROCEDURA PROC    FAR               ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS                ;salva DS sullo stack
          SUB     AX,AX             ;azzera AX
          PUSH    AX                ;salva 0 sullo stack
```

```
         MOV    AX,DATI              ;indirizzo di DATI in AX
         MOV    DS,AX                ;indirizzo di DATI in DS

;codice che esegue la moltiplicazione di due numeri di 32 bit
         MOV    EAX,NUM1             ;carica il primo numero di 32 bit in EAX
         MUL    EAX,NUM2             ;moltiplica con il secondo numero di 32 bit
         MOV    DWORD PTR RISPOSTA,EAX   ;i 32 LSB del risultato in RISPOSTA
         MOV    DWORD PTR RISPOSTA+4,EDX ;i 32 MSB del risultato in RISPOSTA
;fine dell'esempio di moltiplicazione

         RET                         ;il controllo ritorna al DOS
PROCEDURA ENDP                       ;fine della procedura
CODICE   ENDS                        ;fine del segmento di codice

         END                         ;fine del programma
```

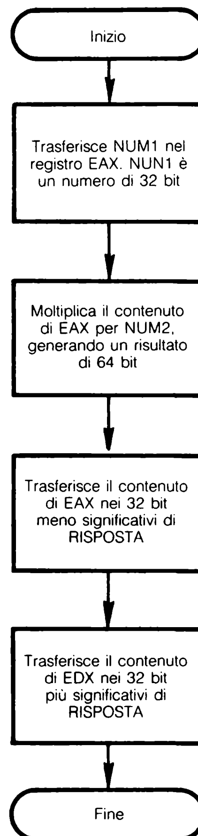


Figura 5.18 Passi del programma che realizza la moltiplicazione sul microprocessore 80386

Il segmento dati del programma è il seguente:

```
DATI        SEGMENT   PARA    'DATI'
NUM1        DD        12345678H
NUM2        DD        9ABCDEF0H
RISPOSTA    DQ        ?
DATI        ENDS
```

In questo esempio, NUM1 e NUM2 sono stati definiti di tipo doubleword, per cui occupano una zona di memoria di 32 bit ciascuno, mentre la variabile RISPOSTA è di tipo quadword, cioè ha una dimensione di 64 bit.
Le istruzioni che costituiscono il nucleo del programma sono molto semplici:

```
MOV  EAX,NUM1                    ;carica il primo numero di 32 bit in EAX
MUL  EAX,NUM2                    ;moltiplica con il secondo numero di 32 bit
MOV  DWORD PTR RISPOSTA,EAX      ;i 32 LSB del risultato in RISPOSTA
MOV  DWORD PTR RISPOSTA+4,EDX    ;i 32 MSB del risultato in RISPOSTA
```

Il primo numero (NUM1) viene trasferito nel registro a 32 bit EAX e viene poi moltiplicato per il secondo numero (NUM2); il risultato di questa operazione viene memorizzato nel registro EDX (i 32 bit più significativi) e nel registro EAX (i 32 bit meno significativi). Utilizzando l'operatore DWORD PTR, il contenuto di questi due registri viene trasferito correttamente negli otto byte di memoria referenziati dalla variabile RISPOSTA, che contengono così il numero esadecimale 0B00EA4E242D2080H, cioè 792891155752493184 in decimale.

5.5 Interruzioni BIOS e DOS

Se il microprocessore è il cuore del calcolatore, le routine BIOS e DOS costituiscono il cervello della macchina. Ogni calcolatore – prima di essere commercializzato – viene programmato con una informazione che risiede nella memoria a sola lettura (ROM: Read Only Memory). La maggior parte delle routine BIOS (Basic Input/Output System) sono allocate permanentemente in questa memoria, mentre il sistema operativo DOS (Disk Operating System) risiede sul disco rigido del calcolatore e viene caricato nella memoria a lettura e scrittura (RAM) in un secondo momento, ogni volta che il calcolatore viene acceso o quando viene eseguita su di esso una operazione di reset.
Il linguaggio assemblatore garantisce l'interfaccia con le routine di sistema, per cui i programmi risultano più efficienti e vengono sviluppati più rapidamente. Ogni microprocessore (dall'8088 fino agli 80286 e 80386) possiede un set di comandi BIOS e DOS per invocare le routine di sistema (l'uso e il si-

gnificato di questi comandi viene generalmente spiegato in dettaglio nella documentazione tecnica di ogni specifica macchina).
I restanti programmi di questo capitolo possono essere eseguiti su calcolatori IBM, che dispongano di tutto il software di supporto necessario, e parte di essi possono essere eseguiti – senza modifiche – anche su calcolatori IBM compatibili, mentre altri richiedono alcuni adattamenti che dipendono dal grado di compatibilità esistente.

INTERRUZIONI BIOS PER CONTROLLARE L'IMMAGINE SULLO SCHERMO

In molti casi, è necessario visualizzare sullo schermo i dati di ingresso e i risultati che vengono prodotti dalle elaborazioni. Per soddisfare questa richiesta, è spesso indispensabile cancellare l'immagine precedentemente apparsa sullo schermo, mentre altre volte al programmatore può interessare cambiare i colori sullo schermo oppure il formato di visualizzazione.
È possibile ottenere facilmente il controllo dell'immagine sullo schermo invocando alcune routine di sistema mediante l'istruzione di chiamata di interruzione BIOS INT 10H. La Tabella 5.2 elenca le numerose funzioni che si possono richiedere da programma per avere pieno controllo dell'immagine sullo schermo.
La chiamata sincrona di interruzione viene indicata con un valore esadecimale (ad esempio, INT 10H), mentre i numeri che controllano il tipo di azioni richieste sono quasi sempre espressi nel formato decimale (ad esempio, MOV AH,13).
Una delle funzioni più utilizzate è quella che permette di cancellare lo schermo prima di continuare l'esecuzione del programma. Questa funzione ha lo stesso effetto del comando DOS CLS (CLear Screen), ma può essere utilizzata a livello di linguaggio assemblatore.
I prossimi due brevi programmi eseguono la cancellazione dello schermo mediante, rispettivamente, la chiamata di interruzione 10H ad una routine BIOS e l'invio di caratteri spazio (blank) direttamente sullo schermo.

```
;per macchine 8088/80286
;programma che pulisce lo schermo invocando un interruzione BIOS

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
STACK     ENDS

CODICE    SEGMENT PARA 'CODICE'     ;definisce il segmento di codice
PROCEDURA PROC    FAR               ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS                ;salva DS sullo stack
          SUB     AX,AX             ;azzera AX
          PUSH    AX                ;salva 0 sullo stack
```

Tabella 5.2 Controllo dell'immagine sullo schermo mediante chiamate di interruzione INT 10H alle routine BIOS sul calcolatore IBM AT (80286) e compatibili. Per le specifiche E.G.A. si veda l'elenco delle routine BIOS (a partire dal 2 Agosto 1984)

Sintassi: INT 10H (parametri già inizializzati al valore richiesto)

Valore AH	**Funzione**	**Ingresso**	**Uscita**
	Controllo dell'interfaccia del video		
AH = 0	Modalità di visualizzazione	AL = 0	40 × 25 b/n
		AL = 1	40 × 25 colore
		AL = 2	80 × 25 b/n
		AL = 3	80 × 25 colore
		AL = 4	320 × 200 grafica colore
		AL = 5	320 × 200 grafica b/n
		AL = 6	640 × 200 grafica b/n
		AL = 10	640 × 350 grafica EGA
AH = 1	Tipo di cursore	CH =	Bit 4 – 0 inizio di linea per cursore
		CL =	Bit 4 – 0 fine di linea per cursore
AH = 2	Posizione del cursore	DH =	Riga
		DL =	Colonna
		BH =	Numero di pagina di visualizzazione
AH = 3	Lettura della posizione del cursore (valori in esecuzione)	DH =	Riga
		DL =	Colonna
		CH =	Modo cursore
		CL =	Modo cursore
		BH =	Numero di pagina di visualizzazione
AH = 4	Posizione penna ottica (valori in esecuzione)	AH = 0	Reset
		AH = 1	Valori validi; prosegue:
		DH =	Riga
		DL =	Colonna
		CH =	Linea grafica (0 – 199)
		BX =	Colonna grafica (0 – 319/639)
AH = 5	Pagina attiva da visualizzare	AL =	Nuovo valore di pagina
			Modi 0 e 1 (0 – 7)
			Modi 2 e 3 (0 – 3)
AH = 6	Aggiornamento verso l'alto della finestra	AL =	Numero di linee; 0 per l'intero schermo
		CH =	Riga, angolo a sinistra in alto
		CL =	Colonna, angolo a sinistra in alto
		DH =	Riga, angolo a destra in basso
		DL =	Colonna, angolo a destra in basso
		BH =	Attributo da usare

Tabella 5.2 (continua)

Sintassi: INT 10H (parametri già inizializzati al valore richiesto)			
Valore AH	**Funzione**	**Ingresso**	**Uscita**
AH = 7	Aggiornamento verso il basso della finestra	AL =	Numero di linee; 0 per l'intero schermo
		CH =	Riga, angolo a sinistra in alto
		CL =	Colonna, angolo a sinistra in alto
		DH =	Riga, angolo a destra in basso
		DL =	Colonna, angolo a destra in basso
		BH =	Attributo da usare
	Manipolazione dei caratteri		
AH = 8	Lettura attributo/carattere alla posizione del cursore	BH =	Visualizza la pagina
		AL =	Lettura carattere
		AH =	Attributo di carattere
AH = 9	Scrittura attributo/carattere alla posizione del cursore	BH =	Visualizza la pagina
		CX =	Numero di caratteri da scrivere
		AL =	Carattere da scrivere
		BL =	Attributo di carattere
AH = 10	Scrittura carattere alla posizione del cursore	BH =	Visualizza la pagina
		CX =	Numero di caratteri da scrivere
		AL =	Carattere da scrivere
	Interfaccia per la grafica		
AH = 11	Selezione del set di colori	BH =	ID Palette (0 – 127)
		BL =	Colore in Palette 0 – Sfondo (0 – 15) 1 – Palette 0 – Verde (1), Rosso (2), Giallo (3) 1 – Cyan (1), Magenta (2), Bianco (3)
AH = 12	Visualizzazione di punti sullo schermo	DX =	Riga (0 – 199)
		CX =	Colonna (0 – 319/639)
		AL =	Colore del punto
AH = 13	Lettura dell'informazione associata al punto	DX =	Riga (0 – 199)
		CX =	Colonna (0 – 319/639)
		AL =	Valore del punto
	Uscita ASCII su telescrivente		
AH = 14	Scrittura in pagina attiva	AL =	Carattere da scrivere
		BL =	Colore dell'immagine in primo piano

Tabella 5.2 (continua)

Sintassi: INT 10H (parametri già inizializzati al valore richiesto)

Valore AH	Funzione	Ingresso	Uscita
AH = 15	Ripristino dello stato del video	AL =	Modalità corrente
		AH =	Numero di colonne sullo shermo
		BH =	Pagina corrente visualizzata
AH = 16	Riservato		
AH = 17	Riservato		
AH = 18	Riservato		
AH = 19	Scrittura stringa	ES:BP =	Punta alla stringa
		CX =	Lunghezza della stringa
		DX =	Posizione INZIALE del cursore
		BH =	Numero di pagina
		AL = 0	BL = attributo (*char,char,char...char*) Il cursore non si muove
		AL = 1	BL = attributo (*char,char,char...char*) Il cursore si muove
		AL = 2	(*char,attr,char,attr...*) Il cursore non si muove
		AL = 3	(*char,attr,char,attr...*) Il cursore si muove

```
;codice che esegue la cancellazione dello schermo
          MOV     CX,0000         ;riga,colonna a sinistra in alto
          MOV     DX,2479H        ;riga,colonna a destra in basso
          MOV     BH,07           ;attributo normale
          MOV     AH,06           ;aggiornamento della finestra in alto
          MOV     AL,00           ;tutto lo schermo
          INT     10H             ;chiamata di interruzione BIOS
;fine dell'esempio di cancellazione dello schermo

          RET                     ;il controllo ritorna al DOS
PROCEDURA ENDP                    ;fine della procedura
CODICE    ENDS                    ;fine del segmento di codice

          END                     ;fine del programma
```

La gestione dell'interruzione è mascherata, nel senso che il programmatore non vede il codice della routine BIOS (scritto dalla IBM, in questo caso) che viene eseguito. Comunque, la IBM ha pubblicato un manuale in cui elenca le routine BIOS disponibili su ogni suo calcolatore. Nella parte di programma qui di seguito riportata, si può facilmente constatare come i registri siano stati inizializzati con valori particolari, dipendenti dal tipo di azione

richiesta (consultare la Tabella 5.2), prima che venga eseguita la chiamata di interruzione.

```
MOV  CX,0000        ;riga,colonna a sinistra in alto
MOV  DX,2479H       ;riga,colonna a destra in basso
MOV  BH,07          ;attributo normale
MOV  AH,06          ;aggiornamento della finestra in alto
MOV  AL,00          ;tutto lo schermo
INT  10H            ;chiamata di interruzione BIOS
```

Il valore 6 nel registro AH specifica che si intende eseguire un "aggiornamento verso l'alto della finestra", mentre se il contenuto di AL è 0 viene cancellato tutto lo schermo. Se BH contiene il valore 7, si inizializza l'attributo di video al valore "normale" (maggiori dettagli sugli attributi vengono forniti successivamente). CX e DX controllano la dimensione della finestra: CH e CL stabiliscono il valore di inizio riga e di inizio colonna (se entrambi i registri sono a 0, la finestra inizia dall'angolo in alto a sinistra dello schermo); DX è stato inizializzato a 2479H, per cui DH contiene il valore esadecimale 24 e DL contiene il valore esadecimale 79 (questo significa che la finestra da cancellare termina in corrispondenza dell'angolo in basso a destra dello schermo). Quando finalmente viene invocata l'interruzione, tutto lo schermo viene cancellato, il cursore ritorna nella posizione che occupava prima dell'esecuzione della routine BIOS e il controllo ripassa al DOS.
È possibile cancellare lo schermo anche inviando su di esso un insieme di caratteri spazio (blank).

```
;per macchine 8088/80286
;programma che cancella lo  schermo senza invocare interruzioni
;BIOS

STACK      SEGMENT PARA STACK
           DB      64 DUP ('MYSTACK ')
STACK      ENDS

CODICE     SEGMENT PARA 'CODICE'      ;definisce il segmento di codice
PROCEDURA PROC     FAR                ;inizio della procedura
           ASSUME  CS:CODICE,DS:DATI,SS:STACK
           PUSH    DS                 ;salva DS sullo stack
           SUB     AX,AX              ;azzera AX
           PUSH    AX                 ;salva 0 sullo stack

;codice che esegue la cancellazione dello schermo a colori
           MOV     AX,0B800H          ;entry-point alla RAM
           MOV     ES,AX              ;ES contiene l'entry-point alla RAM
           MOV     DI,00H             ;indirizzo di partenza
           MOV     AL,00H             ;carattere da scrivere
           MOV     AH,07H             ;attributo di schermo di tipo normale
           MOV     CX,7D0H            ;200 trasferimenti di carattere
           REP     STOSW              ;ripete
;fine dell'esempio di cancellazione dello schermo

           RET                        ;il controllo ritorna al DOS
```

```
PROCEDURA ENDP                          ;fine della procedura
CODICE    ENDS                          ;fine del segmento di codice

          END                           ;fine del programma
```

Questo programma può essere eseguito solo su calcolatori che dispongono di schermo a colori. Ci sono alcune osservazioni da fare sulle seguenti istruzioni:

```
MOV AX,0B800H      ;entry-point alla RAM
MOV ES,AX          ;ES contiene l'entry-point alla RAM
MOV DI,00H         ;indirizzo di partenza
MOV AL,00H         ;carattere da scrivere
MOV AH,07H         ;attributo di schermo di tipo normale
MOV CX,7D0H        ;200 trasferimenti di carattere
REP STOSW          ;ripete
```

L'adattatore di monitor a colori possiede una memoria cui è possibile accedere ponendo il valore 0B800H nel registro ES, mentre per l'adattatore di monitor monocromatici è indispensabile utilizzare il valore numerico 0B000H. Il registro ES non può essere caricato direttamente, ma solo tramite il registro AX. Il registro DI contiene l'offset della locazione corrente di memoria in cui scrivere, il registro AH controlla l'attributo di schermo (il valore 7 indica un attributo normale, cioè schermo bianco, non intermittente e di intensità normale), mentre il registro AL specifica il carattere da scrivere in memoria (in questo caso, il carattere spazio). L'operazione di cancellazione dello schermo viene effettuata trasferendo il carattere spazio (contenuto in AL) in ognuna delle 80 x 25 = 2000 posizioni dello schermo, mediante l'istruzione REP STOSW (REPeat STOre String Word).

INTERRUZIONE BIOS PER VISUALIZZARE UN MESSAGGIO SULLO SCHERMO

È possibile controllare completamente le funzioni di schermo invocando l'interruzione 10H. Il seguente programma permette di visualizzare un messaggio sullo schermo di un monitor a colori.

```
;per macchine IBM 8088/80286 con monitor RGB
;programma che illustra l'uso del comando di visualizzazione di
;stringhe (int 10)

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

DATI    SEGMENT PARA 'DATI'
BACK    DB      2000 DUP (' ')
LOGO    DB      13 DUP (' '),'                                 ',14 DUP (' ')
```

```
        DB      13 DUP (' '),'                                                 ',14 DUP (' ')
        DB      13 DUP (' '),'     Programmazione in linguaggio assemblatore   ',14 DUP (' ')
        DB      13 DUP (' '),'                                                 ',14 DUP (' ')
        DB      13 DUP (' '),'     con la famiglia di processori 80286/80386   ',14 DUP (' ')
        DB      13 DUP (' '),'                                                 ',14 DUP (' ')
        DB      13 DUP (' '),'                                                 ',14 DUP (' ')
DATI    ENDS

CODICE    SEGMENT PARA 'CODICE' ;definisce il segmento di codice
PROCEDURA PROC    FAR           ;inizio della procedura
          ASSUME  CS:CODICE,ES:DATI,SS:STACK
          PUSH    DS            ;salva DS sullo stack
          SUB     AX,AX         ;azzera AX
          PUSH    AX            ;salva 0 sullo stack
          MOV     AX,DATI       ;indirizzo di DATI in AX
          MOV     ES,AX         ;indirizzo di DATI in ES

;il seguente segmento di programma pulisce lo schermo
;visualizzando 80 per 25 caratteri spazio. Se nel registro BL
;memorizziamo il valore 40H (01000000B), tutto lo schermo si
;colora di rosso!
          LEA     BP,BACK       ;stringa di caratteri spazio
          MOV     DX,0000       ;cursore nell'angolo in alto a sinistra
          MOV     AH,19         ;attributo di stringa
          MOV     AL,1          ;stampa di caratteri e aggiornamento cursore
          MOV     BL,01000000B  ;sfondo di colore rosso
          MOV     CX,07D0H      ;visualizza 2000 caratteri spazio
          INT     10H           ;chiamata di interruzione

;il seguente segmento di programma posiziona il cursore nove
;linee dopo quella corrente e visualizza la stringa LOGO. Se nel
;registro BL memorizziamo il valore 4EH (01001110B), l'immagine
;visualizzata si colora di giallo su uno sfondo di color rosso
          LEA     BP,LOGO       ;indirizzo della variabile LOGO
          MOV     DH,09         ;nuova posizione del cursore
          MOV     AH,19         ;attributo di stringa
          MOV     AL,1          ;stampa di caratteri e aggiornamento cursore
          MOV     BL,01001110B  ;immagine gialla su sfondo di colore rosso
          MOV     CX,230H       ;numero di caratteri in LOGO
          INT     10H           ;chiamata di interruzione

          RET                   ;il controllo ritorna al DOS
PROCEDURA ENDP                  ;fine della procedura
CODICE    ENDS                  ;fine del segmento di codice

          END                   ;fine del programma
```

La stringa di caratteri viene visualizzata in giallo su uno sfondo di colore rosso.

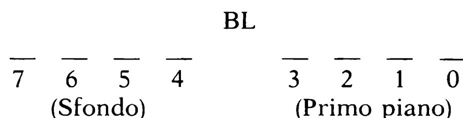
Programmazione in linguaggio assemblatore con la famiglia di processori 80286/80386

Naturalmente è possibile modificare il contenuto del messaggio, facendo però molta attenzione alla spaziatura.

Il programma si compone principalmente di due parti: la prima cancella lo schermo, mentre la seconda visualizza il messaggio. Qui di seguito viene riportata la parte di codice che cancella lo schermo:

```
LEA   BP,BACK       ;stringa di caratteri spazio
MOV   DX,0000       ;cursore nell'angolo in alto a sinistra
MOV   AH,19         ;attributo di stringa
MOV   AL,1          ;stampa di caratteri e aggiornamento cursore
MOV   BL,01000000B  ;sfondo di colore rosso
MOV   CX,07D0H      ;visualizza 2000 caratteri spazio
INT   10H           ;chiamata di interruzione
```

Le routine BIOS disponibili sul calcolatore AT (80286) estendono le funzionalità dell'interruzione 10H, includendo una opzione per la scrittura di stringhe (registro AH inizializzato al valore 19). Il programma utilizza questa opzione due volte: la prima volta per cancellare lo schermo e la seconda volta per visualizzare il messaggio.
Il registro ES referenzia il segmento contenente il messaggio da visualizzare, il cui indirizzo effettivo (BACK) viene caricato nel registro BP, come richiesto dalla routine di gestione dell'interruzione. DX posiziona il cursore nell'angolo in alto a sinistra dello schermo. Se AL contiene il valore 1, la routine di gestione dell'interruzione visualizza un carattere (nel nostro caso un carattere spazio) e aggiorna la posizione del cursore.
Il registro BL permette di controllare il tipo di immagine sullo schermo:



I due termini "sfondo" e "primo piano" rappresentano i colori che sono stati selezionati tra quelli disponibili (Tabella 5.3).
In questa parte di codice, l'immagine sullo schermo viene visualizzata scegliendo il valore 4 (rosso) per colorare lo sfondo e il valore 0 (nessun colore)

Tabella 5.3 Gamma di colori disponibili su PC e AT IBM

0 – Nessun colore	1 – Blue	2 – Grigio
3 – Cyan	4 – Rosso	5 – Magenta
6 – Marrone	7 – Bianco	8 – Grigio
9 – Blue chiaro	10 – Grigio chiaro	11 – Cyan chiaro
12 – Rosso chiaro	13 – Magenta chiaro	14 – Giallo
15 – Bianco scuro		

per visualizzare il messaggio. Il programma tratta direttamente i valori numerici nel formato binario (40H diventa 01000000B, mentre se si vuole avere uno sfondo di colore giallo è necessario memorizzare nel registro BL il valore 0E0H, cioè 11100000B in binario). Nel nostro caso, vengono visualizzati 2000 caratteri spazio su uno sfondo di colore rosso.
La seconda parte di codice visualizza un messaggio sullo schermo. In particolare, i quattro bit più significativi del registro BL contengono l'informazione necessaria per colorare lo sfondo di rosso, mentre i quattro bit meno significativi definiscono di quale colore debba risultare il messaggio (giallo, nel nostro caso), a partire dalla posizione 09 del cursore (registro DH) e per una lunghezza complessiva di 560 caratteri (variabile LOGO).

```
LEA   BP,LOGO        ;indirizzo della variabile LOGO
MOV   DH,09          ;nuova posizione del cursore
MOV   AH,19          ;attributo di stringa
MOV   AL,1           ;stampa di caratteri e aggiornamento cursore
MOV   BL,01001110B   ;immagine gialla su sfondo di colore rosso
MOV   CX,230H        ;numero di caratteri in LOGO
INT   10H            ;chiamata di interruzione
```

INTERRUZIONE BIOS PER VISUALIZZARE DATI SULLO SCHERMO

Mediante l'interruzione DOS 21H (la Tabella 5.4 ne elenca le specifiche di utilizzo) è possibile invocare una particolare funzione di sistema. Il prossimo programma illustra l'impiego di questa interruzione per visualizzare il contenuto di una locazione di memoria, senza ricorrere alla chiamata esplicita di un programma di debug (al contrario di quanto abbiamo fatto nei programmi precedenti).
La Figura 5.19 mostra un programma, la cui utilità può essere verificata esaminando il seguente semplice problema aritmetico:

```
MOV   AX,01234H      ;primo numero
ADD   AX,02299H      ;secondo numero
MOV   TEMPNUM,AX     ;memorizza la somma nella locazione TEMPNUM
```

Il risultato della somma (34CDH) viene memorizzato nella variabile TEMP. Per visualizzare sullo schermo il numero contenuto in TEMP, è indispensabile prima convertirlo in una stringa ASCII, come indicato qui di seguito:

```
          MOV   CX,04H         ;numero di cifre da convertire
ANCORA:   MOV   AX,TEMPNUM     ;i 16 bit del dato sono in AX
          AND   AX,0FH         ;isola i 4 bit meno significativi
          ADD   AL,30H         ;conversione in ASCII
          CMP   AL,39H         ;è una lettera?
```

Tabella 5.4 Specifiche di utilizzo dell'interruzione DOS INT 21H

Sintassi: INT 21H (parametri già inizializzati al valore richiesto)

Valore AH	Funzione	Ingresso	Uscita
AH = 1	Attesa e visualizzazione di carattere da tastiera con controllo CTRL-BREAK in ingresso	=	AL – carattere
AH = 2	Visualizzazione di carattere con controllo CTRL-BREAK	DL =	Carattere da visualizzare
AH = 3	Ingresso asincrono di carattere		AL – carattere in ingresso
AH = 4	Uscita asincrona di carattere	DL =	Carattere da trasferire
AH = 5	Carattere da scrivere	DL =	Carattere da scrivere
AH = 6	Carattere di ingresso da tastiera	DL = 0FFH	Carattere in ingresso; 0 se manca
AH = 7	Attesa di un carattere da tastiera (nessuna visualizzazione)		AL – carattere in ingresso
AH = 8	Attesa di un carattere da tastiera (nessuna visualizzazione – controllo CTRL-BREAK)		AL – carattere in ingresso
AH = 9	Visualizzazione stringa	DS:DX =	Indirizzo di stringa, deve terminare con la sentinella $
AH = A	Stringa da tastiera a buffer	DS:DX =	Indirizzo del buffer. Primo byte = dimensione; secondo byte = numero di caratteri letti
AH = B	Stato della tastiera in ingresso		AL – nessun carattere = 0FFH Carattere = 0
AH = C	Pulitura buffer di tastiera e chiamata di funzione	AL =	1,6,7,8,0A – numero di funzione
AH = D	Drive selezionato (reset)	Nessuno	Nessuna
AH = E	Drive selezionato (selezione)	DL =	AL – numero del drive 0 – drive A 1 – drive B, ecc.
AH = 19	Codice drive (default)		AL – 0 – drive A 1 – drive B, ecc.
AH = 25	Interruzione (attiva)	DS:DX =	Indirizzo del vettore di interruzione
		AL =	Numero di interruzione

Tabella 5.4 continua

Sintassi: INT 10H (parametri già inizializzati al valore richiesto)			
Valore AH	**Funzione**	**Ingresso**	**Uscita**
AH = 2A	Data (lettura)		CX – anno (80 – 90) DH – mese (1 – 12) DL – giorno (1 – 31)
AH = 2B	Data (attiva)	CX:DX	Come sopra AL – 0 se valida OFF se non valida
AH = 2C	Ora (lettura)		CH – ore (0 – 23) CL – minuti (0 – 59)
AH = 2D	Ora (attiva)	CX:DX	Come sopra AL – 0 se valida OFF se non valida
AH = 2E	Stato di verifica (attivo)	DL = AL =	0 0: verifica off 1: verifica on
AH = 35	Indirizzo interruzione (attivo)	AL =	Numero di interruzione ES:BX – punta all'indirizzo del vettore
AH = 36	Disponibilità di spazio su disco	DL =	Drive (0 – default, 1 – A, 2 – B, ecc.) AX – settori/blocco (FFFF se non valido) BX – numero di blocchi liberi CX – byte per settore DX – numero totale di blocchi
AH = 39	Presenza direttorio	DS:DX =	Indirizzo di stringa per il direttorio
AH = 3A	Assenza direttorio	DS:DX =	Indirizzo di stringa per il direttorio
AH = 3B	Modifica direttorio	DS:DX =	Indirizzo di stringa per il nuovo direttorio
AH = 3C	File (creazione)	DS:DX = CX =	Indirizzo di stringa per il file AX – descrittore di file Attributo di file
AH = 3D	File (apertura)	DS:DX = AL =	Indirizzo di stringa per il file 0 – apertura per lettura 1 – apertura per scrittura 2 – apertura per entrambe le operazioni AX – restituisce il descrittore di file

Tabelle 5.4 (continua)

Sintassi: INT 10H (parametri già inizializzati al valore richiesto)			
Valore AH	**Funzione**	**Ingresso**	**Uscita**
AH = 3E	Descrittore file (chiusura)	BX =	Descrittore di file
AH = 3F	File o periferica (lettura)	BX =	Descrittore file
		CX =	Numero di byte da leggere
		DS:DX =	Indirizzo del buffer
			AX – numero di byte letti
AH = 40	File o periferica (scrittura)	BX =	Descrittore file
		CX =	Numero di byte da scrivere
		DS:DX =	Indirizzo del dato da scrivere
			AX – numero di byte scritti
AH = 41	File (cancellazione)	DS:DX =	Indirizzo della stringa del file
AH = 42	File (richiamo attributi)	CX:DX	Offset in byte
		AL = 0	Puntatore ai byte trasferiti (CX:DX) dall'inizio del file
		AL = 1	Puntatore alla locazione corrente più l'offset
		AL = 2	Puntatore all'EOF più l'offset
AH = 43	File (definizione attributi)	DS:DX =	Indirizzo della stringa del file
			Se AL=0, viene restituito l'attributo in CX
			Se AL=1, al file viene assegnato l'attributo in CX
AH = 47	Direttorio corrente	DL =	Numero del drive (0 – default, 1 – drive A, 2 – drive B)
		DS:SI =	Indirizzo del buffer
			DS:SI – restituisce l'indirizzo della stringa
AH = 54	Stato di verifica	Nessuno	AL – 0 se la verifica è off 1 se la verifica è on
AH = 56	File (ridenominazione)	DS:DX =	Indirizzo della stringa per vecchia informazione
		ES:DI =	Indirizzo della stringa per nuova informazione

```
         JL     ASCII             ;se è un numero, salta a ASCII
         ADD    AL,07H            ;conversione della lettera in ASCII
ASCII:   MOV    SI,CX             ;indice uguale al carattere
         MOV    TEMPCHAR[SI],AL   ;salva il numero convertito in carattere
         ROR    TEMPNUM,1         ;rotazione della prossima cifra nella
                                  ;posizione LSB
         ROR    TEMPNUM,1
         ROR    TEMPNUM,1
         ROR    TEMPNUM,1
         ROR    TEMPNUM,1
         LOOP   ANCORA            ;se CX>0, esamina la prossima cifra
```

TEMP è una locazione di 16 bit che contiene quattro cifre esadecimali. Ogni cifra deve essere convertita separatamente nella codifica ASCII equivalente. Innanzitutto, il contenuto di TEMPNUM viene trasferito nel registro AX e viene eseguita l'operazione di prodotto logico tra il contenuto di AX e il valore numerico 0FH, in modo da mantenere solamente le cifre meno significative del numero (in questo caso, 0DH). Sul risultato così ottenuto viene eseguita la somma logica con il valore 30H per ottenere l'equivalente ASCII, seguendo un procedimento distinto a seconda che la cifra corrente rappresenti una lettera oppure un numero: se si tratta di una lettera, infatti, viene aggiunto il valore numerico 07H al contenuto del registro AL (ad esempio, nel caso della lettera D, il registro AL contiene la rappresentazione ASCII 44H). La codifica ASCII della cifra corrente viene memorizzata successivamente nella variabile di tipo stringa TEMPCHAR in corrispondenza della posizione che la cifra stessa occupava nel numero originale. Il contenuto di TEMPNUM viene poi ruotato di quattro bit – cioè una cifra – per cui la cifra corrente viene trasferita nella posizione meno significativa e il processo di decodifica continua fino a quando tutte le quattro cifre sono state convertire nei caratteri ASCII equivalenti.
La parte di codice qui di seguito riportata permette di visualizzare sullo schermo il contenuto di TEMPCHAR:

```
LEA    DX,TEMPCHAR    ;routine DOS di visualizzazione di stringhe
MOV    AH,9           ;parametro DOS
INT    21H            ;interruzione DOS
```

L'istruzione di chiamata di interruzione DOS INT 21H permette di visualizzare i dati memorizzati in un segmento di memoria fino a quando non si incontra il "tappo" – cioè il simbolo $ – che delimita la fine della stringa di caratteri. Se il programmatore dimentica di inserire il tappo, INT 21H continua a visualizzare caratteri fino a quando non incontra il primo carattere $ presente nel codice.

```
;per macchine IBM 8088/80286
;programma che esegue la conversione da esadecimale a ASCII in
;modo da visualizzare il contenuto di un registro sullo schermo

STACK     SEGMENT PARA STACK
          DB     64 DUP ('MYSTACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
TEMPNUM   DW     ?                    ;locazione da visualizzare
TEMPCHAR  DB     8 DUP (' '),'$'      ;stringa di caratteri
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'       ;definisce il segmento di codice
PROCEDURA PROC    FAR                 ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS                  ;salva DS sullo stack
          SUB     AX,AX               ;azzera AX
          PUSH    AX                  ;salva 0 sullo stack
          MOV     AX,DATI             ;indirizzo di DATI in AX
          MOV     DS,AX               ;indirizzo di DATI in DS

;un semplice esempio di somma che illustra l'utilità del
;programma. Possiamo analizzare il risultato della somma, senza
;ricorrere a programmi di debug!
          MOV     AX,01234H           ;primo numero
          ADD     AX,02299H           ;secondo numero
          MOV     TEMPNUM,AX          ;memorizza la somma nella locazione TEMPNUM

;segmento di programma che visualizza sullo schermo il contenuto
;di un registro di 16 bit
          MOV     CX,04H              ;numero di cifre da convertire
ANCORA:   MOV     AX,TEMPNUM          ;i 16 bit del dato sono in AX
          AND     AX,0FH              ;isola i 4 bit meno significativi
          ADD     AL,30H              ;conversione in ASCII
          CMP     AL,39H              ;è una lettera?
          JL      ASCII               ;se è un numero, salta a ASCII
          ADD     AL,07H              ;conversione della lettera in ASCII
ASCII:    MOV     SI,CX               ;indice uguale al carattere
          MOV     TEMPCHAR[SI],AL     ;salva il numero convertito in carattere
          ROR     TEMPNUM,1           ;rotazione della prossima cifra nella posizione LSB
          ROR     TEMPNUM,1
          ROR     TEMPNUM,1
          ROR     TEMPNUM,1
          ROR     TEMPNUM,1
          LOOP    ANCORA              ;se CX>0, esamina la prossima cifra

;routine di visualizzazione di una stringa selezionata in
;corrispondenza della posizione corrente del cursore
          LEA     DX,TEMPCHAR         ;routine DOS di visualizzazione di stringhe
          MOV     AH,9                ;parametro DOS
          INT     21H                 ;interruzione DOS

          RET                         ;il controllo ritorna al DOS
PROCEDURA ENDP                        ;fine della procedura
CODICE    ENDS                        ;fine del segmento di codice

          END                         ;fine del programma
```

Figura 5.19 Visualizzazione del contenuto di un registro invocando l'interruzione DOS 21H

INTERRUZIONE DOS PER LEGGERE UN CARATTERE DA TASTIERA

Spesso è necessario realizzare un programma interattivo, cioè un programma che richiede all'utente di rispondere da tastiera – affermativamente (S) o negativamente (N) – ad una domanda formulata su video. Il seguente programma invoca un'interruzione DOS per richiedere informazioni da tastiera:

```
;per macchine IBM 8088/80286
;programma che esegue la lettura e la visualizzazione con eco di
;un carattere inviato da tastiera. Se il carattere è "S" o "N"
;viene caricato un particolare valore nella variabile di nome
;"test"

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
TEST      DW      ?
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'    ;definisce il segmento di codice
PROCEDURA PROC    FAR              ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS               ;salva DS sullo stack
          SUB     AX,AX            ;azzera AX
          PUSH    AX               ;salva 0 sullo stack
          MOV     AX,DATI          ;indirizzo di DATI in AX
          MOV     DS,AX            ;indirizzo di DATI in DS

;codice che richiede un carattere da tastiera e lo visualizza
;sullo schermo
          MOV     AH,01H           ;parametro per la lettura di un carattere
          INT     21H              ;visualizza un carattere
          CMP     AL,'S'           ;si tratta di una "S"?
          JNE     QUI              ;se no, salta a QUI
          MOV     BX,9999H         ;se si, trasferisce 9999H in BX
          JMP     FINE             ;e finisce il programma
QUI:      CMP     AL,'N'           ;si tratta di una "N"?
          JNE     FINE             ;se no, salta a FINE
          MOV     BX,5555H         ;se si, trasferisce 9999H in BX

FINE:     MOV     TEST,BX          ;salva il valore in TEST

          RET                      ;il controllo ritorna al DOS
PROCEDURA ENDP                     ;fine della procedura
CODICE    ENDS                     ;fine del segmento di codice

          END                      ;fine del programma
```

Le azioni eseguite da questo programma dipendono dal tipo di risposta che l'utente inserisce da tastiera (S oppure N. Si tenga presente che le lettere minuscole s e n non vengono interpretate correttamente dal programma). Riportiamo qui di seguito le istruzioni di programma di cui è conveniente fornire una spiegazione dettagliata:

```
MOV    AH,01H    ;parametro per la lettura di un carattere
INT    21H       ;visualizza un carattere
```

```
        CMP     AL,'S'      ;si tratta di una "S"?
        JNE     QUI         ;se no, salta a QUI
        MOV     BX,9999H    ;se sì, trasferisce 9999H in BX
        JMP     FINE        ;e finisce il programma
QUI:    CMP     AL,'N'      ;si tratta di una "N"?
        JNE     FINE        ;se no, salta a FINE
        MOV     BX,5555H    ;se sì, trasferisce 9999H in BX
FINE:   MOV     TEST,BX     ;salva il valore in TEST
```

La Tabella 5.4 elenca le specifiche che sono necessarie per invocare correttamente le interruzioni DOS INT 21H. Quando il registro AH contiene il valore 1 e viene eseguita la chiamata di interruzione, il programma si pone in attesa di informazioni da tastiera. In particolare, il carattere inserito da tastiera viene visualizzato con eco sullo schermo e nel registro AL viene memorizzata la codifica ASCII equivalente al carattere stesso. Se l'utente ha inserito la lettera S, il programma memorizza nel registro BX – ed eventualmente nella variabile TEST – il valore 9999H; se invece il carattere in ingresso è N, il programma memorizza il valore 5555H, mentre se è stato premuto un tasto diverso dai due precedenti, il contenuto originale di BX viene memorizzato nella variabile TEST.

INTERRUZIONE DOS PER LEGGERE UNA STRINGA DA TASTIERA

Per leggere da tastiera – e memorizzare – una stringa di caratteri, è indispensabile prendere alcune decisioni prima di invocare l'interruzione DOS INT 21H. Consideriamo il seguente segmento dati:

```
BUFF    DB   80 DUP (' '),'$'   ;buffer per la stringa
LINEA   DB   0AH,0DH,'$'        ;ritorno a capo e avanzamento linea
```

La variabile BUFF è in grado di memorizzare una stringa di 80 caratteri, che termina con il carattere tappo $. La variabile LINEA contiene due caratteri di controllo: il primo rappresenta il ritorno a capo, mentre il secondo l'avanzamento linea e permettono di aggiornare la posizione del cursore. Consideriamo il programma che esegue la lettura di una stringa da tastiera:

```
;per macchine IBM 8088/80286
;programma che legge da tastiera e visualizza con eco  una
;stringa di caratteri

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

DATI    SEGMENT PARA 'DATI'
BUFF    DB      80 DUP (' '),'$'  ;buffer per la stringa
LINEA   DB      0AH,0DH,'$'       ;ritorno a capo e avanzamento linea
DATI    ENDS
```

```
CODICE    SEGMENT PARA 'CODICE'    ;definisce il segmento di codice
PROCEDURA PROC    FAR              ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS               ;salva DS sullo stack
          SUB     AX,AX            ;azzera AX
          PUSH    AX               ;salva 0 sullo stack
          MOV     AX,DATI          ;indirizzo di DATI in AX
          MOV     DS,AX            ;indirizzo di DATI in DS
          LEA     BX,BUFF          ;indirizzo di BUFF in BX

;codice che esegue la lettura di una stringa da tastiera e la
;visualizzazione sullo schermo
          MOV     AH,01H           ;parametro per la lettura di carattere
          MOV     CX,00H           ;contatore a 0
QUI:      INT     21H              ;visualizza un carattere
          CMP     AL,0DH           ;è un ritorno a capo?
          JE      BASTA            ;se si, fine della routine
          MOV     BUFF[BX],AL      ;il carattere letto è in memoria
          CMP     CX,79            ;è l'ottantesimo carattere?
          JE      BASTA            ;se si, fine della routine
          INC     CX               ;incrementa il contatore dei caratteri
          INC     BX               ;aggiorna l'indice
          JMP     QUI              ;altro carattere

;codice che visualizza la stringa di caratteri che è stata
;precedentemente memorizzata
BASTA:
          LEA     DX,LINEA         ;avanzamento linea e ritorno a capo
          MOV     AH,09            ;prima di visualizzare la stringa
          INT     21H              ;interruzione DOS
          LEA     DX,BUFF          ;visualizza la stringa di caratteri
          MOV     AH,09
          INT     21H              ;interruzione DOS

          RET                      ;il controllo ritorna al DOS
PROCEDURA ENDP                     ;fine della procedura
CODICE    ENDS                     ;fine del segmento di codice

          END                      ;fine del programma
```

In questo programma, abbiamo utilizzato un contatore per definire il numero corrente di caratteri che vengono inseriti da tastiera. Un altro modo per indicare la conclusione dell'inserimento di caratteri da tastiera è quello di premere il tasto di ritorno a capo (0DH). La dimensione della stringa può variare da 1 a 80 caratteri. Riportiamo qui di seguito le istruzioni del programma che realizzano l'acquisizione dati da tastiera:

```
        MOV   AH,01H        ;parametro per la lettura di carattere
        MOV   CX,00H        ;contatore a 0
QUI:    INT   21H           ;visualizza un carattere
        CMP   AL,0DH        ;è un ritorno a capo?
        JE    BASTA         ;se sì, fine della routine
        MOV   BUFF[BX],AL   ;il carattere letto è in memoria
        CMP   CX,79         ;è l'ottantesimo carattere?
        JE    BASTA         ;se sì, fine della routine
        INC   CX            ;incrementa il contatore dei caratteri
```

```
INC     BX          ;aggiorna l'indice
JMP     QUI         ;altro carattere
```

La lettura dei caratteri da tastiera viene effettuata in maniera simile a quanto abbiamo descritto nel precedente programma. Il contenuto del registro AL (carattere ASCII) viene confrontato con 0DH per verificare se l'informazione in ingresso coincide con il carattere di ritorno a capo. In caso di verifica positiva, il comando JE permette di uscire dal ciclo di acquisizione dati da tastiera; se la verifica dà esito negativo, il carattere viene memorizzato nella variabile BUFF in corrispondenza della locazione di memoria puntata dal contenuto corrente del registro BX. In seguito viene eseguita un'altra verifica per controllare se il numero di caratteri letti ha raggiunto il valore 80 e, in caso di esito negativo, vengono incrementati i due registri CX e BX. CX memorizza il numero di caratteri letti, mentre BX contiene il puntatore alla locazione corrente di memoria – interna alla variabile BUFF – che è destinata a contenere il prossimo carattere letto. Il ciclo viene ripetuto fino a quando il carattere letto equivale al ritorno a capo oppure il numero di caratteri letti ha raggiunto quota 80.
L'ultima parte del programma visualizza due stringhe sullo schermo:

```
LEA     DX,LINEA    ;avanzamento linea e ritorno a capo
MOV     AH,09       ;prima di visualizzare la stringa
INT     21H         ;interruzione DOS
LEA     DX,BUFF     ;visualizza la stringa di caratteri
MOV     AH,09
INT     21H         ;interruzione DOS
```

La variabile LINEA contiene i caratteri di avanzamento linea e di ritorno a capo, mentre la variabile BUFF memorizza la stringa che è stata inserita in ingresso da tastiera. Il programma precedente visualizza semplicemente la stringa sullo schermo a partire dalla posizione corrente del cursore.

INTERRUZIONE BIOS PER LEGGERE LA DATA E L'ORA CORRENTI

L'istruzione di chiamata dell'interruzione BIOS 1AH – disponibile sul calcolatore IBM AT (80286) – è la versione aggiornata di quella supportata dalla CPU 8088 e permette la lettura e la scrittura della data e dell'ora correnti riferite all'orologio di sistema. La Tabella 5.5 elenca le funzioni che è possibile richiedere invocando l'interruzione BIOS 1AH. In aggiunta a queste, il programmatore può attivare o meno la funzione di allarme dell'80286. Il seguente programma – eseguibile solo su macchine che dispongono della CPU 80286 – permette la lettura della data e dell'ora dall'orologio di sistema.

```
;per macchine IBM 80286
;programma che esegue la lettura della data e dell'ora correnti
;invocando un'interruzione BIOS
```

```
STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
ORA       DD      OH                 ;doubleword per memorizzare l'ora
DATA      DD      OH                 ;doubleword per memorizzare la data
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'      ;definisce il segmento di codice
PROCEDURA PROC    FAR                ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS                 ;salva DS sullo stack
          SUB     AX,AX              ;azzera AX
          PUSH    AX                 ;salva 0 sullo stack
          MOV     AX,DATI            ;indirizzo di DATI in AX
          MOV     DS,AX              ;indirizzo di DATI in DS

;routine che memorizza l'ora invocando l'interruzione BIOS
          MOV     AH,02              ;parametro per l'ora
          INT     1AH                ;interruzione ora/data
          MOV     BYTE PTR ORA+3,DH  ;secondi in ORA
          MOV     BYTE PTR ORA+2,CL  ;minuti in ORA
          MOV     BYTE PTR ORA+1,CH  ;ore in ORA

;routine che memorizza la data invocando l'interruzione BIOS
          MOV     AH,04              ;parametro per la data
          INT     1AH                ;interruzione ora/data
          MOV     BYTE PTR DATA+3,DL ;giorno in DATA
          MOV     BYTE PTR DATA+2,DH ;mese in DATA
          MOV     BYTE PTR DATA+1,CL ;anno in DATA
          MOV     BYTE PTR DATA,CH   ;secolo in DATA

          RET                        ;il controllo ritorna al DOS
PROCEDURA ENDP                       ;fine della procedura
CODICE    ENDS                       ;fine del segmento di codice

          END                        ;fine del programma
```

Il programma si compone essenzialmente di due parti. Riportiamo qui di seguito la parte di codice che esegue la lettura dell'ora corrente:

```
MOV   AH,02                  ;parametro per l'ora
INT   1AH                    ;interruzione ora/data
MOV   BYTE PTR ORA + 3,DH    ;secondi in ORA
MOV   BYTE PTR ORA + 2,CL    ;minuti in ORA
MOV   BYTE PTR ORA + 1,CH    ;ore in ORA
```

Il registro AH contiene il valore 2, per cui la chiamata di interruzione INT 1AH restituisce l'ora corrente codificata in tre registri di 8 bit. Le precedenti istruzioni memorizzano, attraverso l'uso dell'operatore PTR, l'ora in una variabile di tipo doubleword (DD) e di nome ORA.
La seguente parte di programma invece legge e decodifica la data corrente:

```
MOV   AH,04                  ;parametro per la data
INT   1AH                    ;interruzione ora/data
```

Tabella 5.5 Opzioni dell'interruzione BIOS 1AH disponibili su calcolatore IBM AT (80286)

Sintassi: INT 1AH (parametri già inizializzati al valore richiesto)

Funzione: permette la lettura o l'aggiornamento del clock di sistema

Valore AH	Funzione	Ingresso	Uscita
AH = 0	Lettura clock corrente		CX – byte più significativi del clock DX – byte meno significativi del clock AL – 0, se non si sono oltrepassate le 24 ore
AH = 1	Aggiornamento clock corrente	CX = DX =	byte più significativi del clock byte meno significativi del clock
AH = 2	Lettura ora – real time clock		CH – ore BCD CL – minuti BCD DH – secondi BCD
AH = 3	Aggiornamento ora – real time clock	CH = CL = DH = DL =	ore BCD minuti BCD secondi BCD 1 – ora legale 0 – ora solare
AH = 4	Lettura data – real time clock		CH – secolo BCD CL – anno BCD DH – mese BCD DL – giorno BCD
AH = 5	Aggiornamento data – real time clock	CH = CL = DH = DL =	secolo BCD anno BCD mese BCD giorno BCD
AH = 6	Aggiornamento allarme (fino a 23:59:59)	CH = CL = DH =	ore BCD minuti BCD secondi BCD
AH = 7	Reset allarme		

```
MOV    BYTE PTR DATA+3,DL     ;giorno in DATA
MOV    BYTE PTR DATA+2,DH     ;mese in DATA
MOV    BYTE PTR DATA+1,CL     ;anno in DATA
MOV    BYTE PTR DATA,CH       ;secolo in DATA
```

Perché venga letta la data, è necessario che il registro AH contenga il valore 4. Dopo che è stata eseguita la chiamata dell'interruzione 1AH, quattro regi-

stri di 8 bit contengono l'informazione sulla data corrente. Il programma utilizza ancora l'operatore PTR per memorizzare la data in una variabile di tipo doubleword (DD) e di nome DATA. Il contenuto di entrambe le variabili ORA e DATA può essere visualizzato utilizzando il programma Debug della IBM.

INTERRUZIONE BIOS PER CALCOLARE LA DIMENSIONE DI MEMORIA DELL'AT

È possibile determinare la quantità di memoria disponibile sul calcolatore IBM AT invocando due interruzioni BIOS. Esaminiamo il programma che permette questo calcolo:

```
;per macchine IBM 80286
;programma che esegue il calcolo della quantità di memoria
;disponibile in un IBM AT mediante chiamata di un'interruzione
;BIOS

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
MEMSYS    DW      ?                ;memoria principale (fino a 640K)
MEMEXT    DW      ?                ;estensione di memoria (oltre i 640K)
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'    ;definisce il segmento di codice
PROCEDURA PROC    FAR              ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS               ;salva DS sullo stack
          SUB     AX,AX            ;azzera AX
          PUSH    AX               ;salva 0 sullo stack
          MOV     AX,DATI          ;indirizzo di DATI in AX
          MOV     DS,AX            ;indirizzo di DATI in DS

;routine che calcola la dimensione della memoria principale
          INT     12H              ;chiamata di interruzione per la memoria
          MOV     MEMSYS,AX        ;memorizza la quantità di memoria principale

;routine che calcola l'estensione di memoria
          MOV     AH,88H           ;parametro per l'interruzione
          INT     15H              ;chiamata interruzione per estensione memoria
          MOV     MEMEXT,AX

          RET                      ;il controllo ritorna al DOS
PROCEDURA ENDP                     ;fine della procedura
CODICE    ENDS                     ;fine del segmento di codice

          END                      ;fine del programma
```

In realtà, l'interruzione 12H viene supportata sia dal PC (8088) che dall'AT (80286) e permette la lettura della quantità totale di memoria installata nel sistema.

```
INT     12H           ;chiamata di interruzione per la memoria
MOV     MEMSYS,AX     ;memorizza la quantità di memoria principale
```

Il valore che viene memorizzato in AX corrisponde al numero di blocchi presenti in memoria, aventi la dimensione di 1 kB.
Viene invocata inoltre l'interruzione 15H per calcolare l'estensione di memoria:

```
MOV     AH,88H        ;parametro per l'interruzione
INT     15H           ;chiamata interruzione per estensione memoria
MOV     MEMEXT,AX
```

Sul PC IBM (8088), l'interruzione 15H permette di realizzare funzioni di I/O su cassetta magnetica.
Il calcolatore AT (80286) non dispone di una porta per cassetta magnetica, per cui questa interruzione permette di realizzare altre funzioni, tra cui quella di determinare l'estensione di memoria. Dopo che il registro AX è stato inizializzato al valore 88H ed è stata eseguita la chiamata dell'interruzione, lo stesso registro AX contiene il valore dell'estensione di memoria, in termini di numero di blocchi di 1 kB disponibili, in aggiunta ai 640 kB che vengono garantiti dalla memoria principale.

ATTREZZATURA DI SUPPORTO INSTALLATA SUL CALCOLATORE

Il seguente programma – invocando una interruzione DOS – analizza alcune parti dell'attrezzatura di supporto che sono disponibili sui calcolatori PC e AT.

```
;per macchine IBM 8088/80286
;programma che analizza l'attrezzatura di supporto disponibile su
;un calcolatore IBM, mediante chiamata di interruzione BIOS

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
SUPPORTO  DW      ?
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'    ;definisce il segmento di codice
PROCEDURA PROC    FAR              ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS               ;salva DS sullo stack
          SUB     AX,AX            ;azzera AX
          PUSH    AX               ;salva 0 sullo stack
          MOV     AX,DATI          ;indirizzo di DATI in AX
          MOV     DS,AX            ;indirizzo di DATI in DS
```

```
;routine che determina l'attrezzatura di supporto
        INT     11H             ;interruzione BIOS
        MOV     SUPPORTO,AX     ;SUPPORTO contiene l'informazione richiesta

        RET                     ;il controllo ritorna al DOS
PROCEDURA ENDP                  ;fine della procedura
CODICE  ENDS                    ;fine del segmento di codice

        END                     ;fine del programma
```

Questo programma può essere eseguito su calcolatori PC o AT, ma l'informazione che viene memorizzata – nei due casi – nella variabile SUPPORTO assume un significato diverso. Esaminiamo in particolare le seguenti istruzioni:

```
INT   11H           ;interruzione BIOS
MOV   SUPPORTO,AX   ;SUPPORTO contiene l'informazione richiesta
```

Tabella 5.6 Significato dei singoli bit del registro AX per la determinazione dei dispositivi supportati dal sistema (interruzione 11H)

Bit	Dispositivi supportati
Calcolatori PC e XT	
15 – 14	Numero di stampanti
13	Non utilizzato
12	Adattatore di giochi
11 – 10 – 9	Numero di schede RS-232
8	Non utilizzato
7 – 6	Numero di drive per floppy; se il bit 0=1, 00=1, 01=2, 10=3, 11=4
5 – 4	Modalità di visualizzazione iniziale 01 = 40 x 25 b/n usando la scheda colore 10 = 80 x 25 b/n usando la scheda colore 11 = 80 x 25 b/n usando la scheda monocromatica
3 – 2	Dimensione RAM 00 = 16 kB 01 = 32 kB 10 = 48 kB 11 = 64 kB
1	Non utilizzato
0	Drive per floppy
Calcolatori AT (come sopra, tranne i seguenti)	
12	Non utilizzato
3 – 2	Non utilizzati
1	Coprocessore 80287

Per visualizzare il contenuto della variabile SUPPORTO, è possibile richiedere – dopo l'esecuzione del programma – l'immagine della zona di memoria occupata dal segmento dati oppure aggiungere ulteriori istruzioni che permettano di visualizzare il contenuto del registro AX (si riveda la Figura 5.19). Il valore esadecimale memorizzato in AX deve essere poi convertito nel formato binario in modo che ognuno dei 16 bit possa essere esaminato. La Tabella 5.6 mostra il significato dei diversi bit.
Se, ad esempio, nel registro AX viene memorizzato il valore 4463H, la conversione nel formato binario è la seguente:

0	1	0	0	0	1	0	0	0	1	1	0	0	0	1	1	(numero binario)
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	(posizione dei bit)

La precedente codifica indica che l'attrezzatura di supporto disponibile sul calcolatore, leggendo da sinistra a destra, risulta:

Drive installati
Coprocessore matematico 80287
Modalità di visualizzazione 80×25 b/n con scheda colore
2 – drive per floppy
2 – porte RS-232
1 – Stampante

INTERRUZIONE BIOS PER INVIARE UNA STRINGA DI CARATTERI ALLA STAMPANTE

Il seguente programma permette di inviare una stringa di caratteri alla stampante di sistema (LPT1:).

```
;per macchine IBM 8088/80286
;programma che invia una stringa di caratteri alla stampante
;(LPT1:) mediante chiamata di interruzione BIOS. Premere Ctrl-PrtSc
;prima di iniziare l'esecuzione

STACK      SEGMENT PARA STACK
           DB      64 DUP ('MYSTACK ')
STACK      ENDS

DATI       SEGMENT PARA 'DATI'
MESSAGGIO DB       'La programmazione in assembler è rapida','$'
DATI       ENDS

CODICE     SEGMENT PARA 'CODICE'    ;definisce il segmento di codice
PROCEDURA PROC     FAR              ;inizio della procedura
           ASSUME  CS:CODICE,DS:DATI,SS:STACK
           PUSH    DS               ;salva DS sullo stack
           SUB     AX,AX            ;azzera AX
           PUSH    AX               ;salva 0 sullo stack
           MOV     AX,DATI          ;indirizzo di DATI in AX
```

```
        MOV     DS,AX              ;indirizzo di DATI in DS
        MOV     BX,MESSAGGIO       ;indirizzo di MESSAGGIO in BX

;routine che invia un carattere per volta alla stampante
        MOV     DX,00H             ;indirizzo della stampante
        MOV     AH,01H             ;parametro per inizializzare la stampante

        INT     17H                ;interruzione per la stampa
ANCORA: MOV     AL,MESSAGGIO[BX]   ;prende un carattere dalla stringa
        CMP     AL,'$'             ;fine del messaggio?
        JE      FINE               ;se sì, fine della routine
        MOV     AH,00H             ;parametro per stampa del carattere
        INT     17H                ;interruzione per la stampa
        INC     BX                 ;prossimo carattere da stampare
        JMP     ANCORA             ;ripete

FINE:

        RET                        ;il controllo ritorna al DOS
PROCEDURA ENDP                     ;fine della procedura
CODICE  ENDS                       ;fine del segmento di codice

        END                        ;fine del programma
```

Il comando CTRL-PRTSC deve essere inviato prima dell'esecuzione del programma. Come accade nell'operazione di acquisizione di caratteri da tastiera, anche l'invio dei dati alla stampante deve procedere carattere per carattere. La parte di programma qui di seguito riportata mostra come viene eseguita l'operazione di stampa di un messaggio:

```
         MOV   DX,00H             ;indirizzo della stampante
         MOV   AH,01H             ;parametro per inizializzare
                                  ;la stampante
         INT   17H                ;interruzione per la stampa
ANCORA:  MOV   AL,MESSAGGIO[BX]   ;prende un carattere dalla stringa
         CMP   AL,'$'             ;fine del messaggio?
         JE    FINE               ;se sì, fine della routine
         MOV   AH,00H             ;parametro per stampa del carattere
         INT   17H                ;interruzione per la stampa
         INC   BX                 ;prossimo carattere da stampare
         JMP   ANCORA             ;ripete
FINE:
```

Per poter invocare correttamente l'interruzione 17H, è indispensabile prima inizializzare la stampante. Il contenuto del registro DX (0, 1 oppure 2) definisce l'indirizzo logico della stampante (al valore 0 corrisponde, ad esempio, l'indirizzo logico LPT1), mentre il contenuto del registro AH (valore 1) inizializza la porta della stampante. È sufficiente eseguire queste inizializzazioni una sola volta, prima di invocare l'interruzione 17H.
Il programma entra in ciclo per inviare alla stampante un carattere alla volta, carattere che era stato precedentemente trasferito dalla variabile MESSAGGIO al registro AL. Occorre verificare che il carattere da inviare non

coincida con il tappo, nel qual caso l'operazione di stampa deve essere interrotta. Se, invece, la verifica ha esito negativo, viene memorizzato nel registro AH il valore 0 e la successiva invocazione dell'interruzione 17H permette di stampare il carattere corrente; inoltre, viene incrementato il registro BX, in modo da referenziare il successivo carattere del messaggio e ripetere correttamente il ciclo.

INTERRUZIONE BIOS PER VISUALIZZARE I PUNTI SU UNO SCHERMO A COLORI A MEDIA RISOLUZIONE

La famiglia IBM di personal computer supporta due tipi di grafica su monitor standard a colori: una grafica a media risoluzione e una grafica ad alta risoluzione. La grafica a media risoluzione permette di visualizzare 320 x 200 punti, con al massimo quattro colori contemporaneamente presenti sullo schermo, mentre la grafica ad alta risoluzione permette di visualizzare 640 x 200 punti, con due colori (di solito il bianco e il nero). La Tabella 5.2 – già presentata in questo capitolo – indica che i valori 11, 12 e 13 contenuti nel registro AH definiscono il tipo di interfaccia grafica per il BIOS.
Il seguente programma permette di visualizzare tre punti al centro dello schermo.

```
;per macchine IBM 8088/80286 con monitor RGB
;programma che visualizza tre punti su uno schermo a colori con
;grafica a media risoluzione mediante chiamata di interruzione
;BIOS. Richiede l'uso della scheda colore grafica

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
STACK     ENDS

CODICE    SEGMENT PARA 'CODICE'   ;definisce il segmento di codice
PROCEDURA PROC    FAR             ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS              ;salva DS sullo stack
          SUB     AX,AX           ;azzera AX
          PUSH    AX              ;salva 0 sullo stack

;routine che visualizza tre punti al centro dello schermo
          MOV     AH,00           ;definisce la modalità grafica, cioè
          MOV     AL,04           ;a media risoluzione 320 per 200 a colori
          INT     10H             ;chiamata di interruzione

          MOV     AH,11           ;definisce la gamma di colori
          MOV     BH,00           ;definisce il colore dello sfondo
          MOV     BL,01           ;sfondo blu
          INT     10H             ;chiamata di interruzione

          MOV     AH,11           ;definisce la gamma di colori
          MOV     BH,01           ;seleziona il colore dell'immagine
          MOV     BL,00           ;verde/rosso/giallo
          INT     10H             ;chiamata di interruzione
```

```
          MOV       AL,02          ;punti di colore rosso
          MOV       AH,12          ;parametro per scrittura punto
          MOV       DX,64H         ;centesima riga (verticale)
          MOV       CX,9EH         ;centocinquantottesima colonna (orizzontale)
          INT       10H            ;chiamata di interruzione
          MOV       AH,12          ;parametro per scrittura punto
          MOV       CX,0A0H        ;altro punto sullo schermo
          INT       10H            ;chiamata di interruzione
          MOV       AH,12          ;parametro per scrittura punto
          MOV       CX,0A2H        ;altro punto sullo schermo
          INT       10H            ;chiamata di interruzione

          RET                      ;il controllo ritorna al DOS
PROCEDURA ENDP                     ;fine della procedura
CODICE    ENDS                     ;fine del segmento di codice

          END                      ;fine del programma
```

Prima di visualizzare i punti sullo schermo, è indispensabile inizializzare la grafica del calcolatore alla modalità a media risoluzione, come qui di seguito riportato:

```
MOV   AH,00    ;definisce la modalità grafica, cioè
MOV   AL,04    ;a media risoluzione 320 per 200 a colori
INT   10H      ;chiamata di interruzione
```

Si consulti la Tabella 5.2 per avere maggiori dettagli sul significato dei valori contenuti nei registri AH e AL.
A questo punto, è necessario definire i colori per lo sfondo e per l'immagine sullo schermo:

```
MOV   AH,11    ;definisce la gamma di colori
MOV   BH,00    ;definisce il colore dello sfondo
MOV   BL,01    ;sfondo blu
INT   10H      ;chiamata di interruzione
```

Il registro BH contiene il valore 0, ad indicare che viene definita la colorazione dello sfondo, che può assumere 16 tonalità distinte (numerate da 0 a 15, come indicato nella Tabella 5.2).
Il colore scelto per lo sfondo – in questo caso – è il blu, mentre il colore da assegnare alle immagini visualizzate sullo schermo viene determinato con le seguenti istruzioni:

```
MOV   AH,11    ;definisce la gamma di colori
MOV   BH,01    ;seleziona il colore dell'immagine
MOV   BL,00    ;verde/rosso/giallo
INT   10H      ;chiamata di interruzione
```

Il registro BH contiene il valore 1 ad indicare che si intende selezionare il colore per le immagini da visualizzare sullo schermo, mentre il registro BL specifica la gamma dei colori (verde/rosso/giallo). Questi tre passi di inizia-

lizzazione vengono eseguiti una sola volta, a meno che non si intenda modificare un colore sullo schermo.
La parte restante del programma utilizza tre volte il comando con cui vengono visualizzati i punti sullo schermo:

```
MOV   AL,02     ;punti di colore rosso
MOV   AH,12     ;parametro per scrittura punto
MOV   DX,64H    ;centesima riga (verticale)
MOV   CX,9EH    ;centocinquantottesima colonna (orizzontale)
INT   10H       ;chiamata di interruzione
MOV   AH,12     ;parametro per scrittura punto
MOV   CX,0A0H   ;altro punto sullo schermo
INT   10H       ;chiamata di interruzione
MOV   AH,12     ;parametro per scrittura punto
MOV   CX,0A2H   ;altro punto sullo schermo
INT   10H       ;chiamata di interruzione
```

Quando viene eseguito il comando che permette di visualizzare i punti sullo schermo, il registro CX contiene il numero di colonna, mentre il registro DX contiene il numero di riga. Nel nostro caso, i punti vengono visualizzati in rosso, in quanto nel registro AL abbiamo memorizzato il valore 2. Per la visualizzazione dei restanti due punti, viene modificato opportunamente solo il contenuto del registro CX.
Il BASIC e il Pascal supportano molti comandi di grafica, ma solo il BIOS fornisce i comandi di visualizzazione di punti. Quindi, se il programmatore intende tracciare una linea sullo schermo, deve costruirla punto per punto e utilizzare i comandi che la visualizzino nella direzione voluta. Non ci sono problemi se la linea è verticale od orizzontale, altrimenti sorgono alcune difficoltà (si pensi alle linee curve!). Questo programma può essere eseguito su calcolatori che dispongono di una grafica a colori a media risoluzione.

INTERRUZIONE BIOS PER VISUALIZZARE UNA LINEA SULLO SCHERMO AD ALTA RISOLUZIONE

L'esempio del precedente paragrafo può essere generalizzato, in modo da poter disegnare una linea su uno schermo ad alta risoluzione. Il seguente programma mostra come fare, utilizzando un ciclo.

```
;per macchine IBM 8088/80286 con monitor RGB
;programma che visualizza una linea, come successione di punti,
;sullo schermo  che dispone di una grafica ad alta risoluzione,
;mediante chiamata di interruzione BIOS
;Richiede l'uso della scheda colore grafica

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
STACK     ENDS
```

```
CODICE    SEGMENT PARA 'CODICE'    ;definisce il segmento di codice
PROCEDURA PROC    FAR              ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS               ;salva DS sullo stack
          SUB     AX,AX            ;azzera AX
          PUSH    AX               ;salva 0 sullo stack

;routine che visualizza sullo schermo una linea come successione
;di punti
          MOV     AH,00            ;definisce la modalità grafica, cioè
          MOV     AL,06            ;ad alta risoluzione 640 per 200
          INT     10H              ;chiamata di interruzione

          MOV     AL,01            ;punti di colore bianco o nero
          MOV     DX,00            ;posizione iniziale: in alto
          MOV     CX,00            ;a sinistra
ANCORA:   MOV     AH,12            ;parametro per la scrittura di punti
          INT     10H              ;chiamata di interruzione
          INC     DX               ;incremento della posizione verticale
          INC     CX               ;incremento della posizione orizzontale
          INC     CX
          INC     CX
          CMP     DX,0C8H          ;la linea è al limite dello schermo (200)?
          JE      FINE             ;se si, fine del programma
          JMP     ANCORA           ;se no, si aggiunge alla linea un altro punto

FINE:

          RET                      ;il controllo ritorna al DOS
PROCEDURA ENDP                     ;fine della procedura
CODICE    ENDS                     ;fine del segmento di codice

          END                      ;fine del programma
```

Poiché vengono utilizzati due soli colori (bianco e nero) – cioè quelli di default – non è necessario definire esplicitamente i colori che devono assumere lo sfondo e le immagini sullo schermo. La parte di programma, che viene qui di seguito riportata, indica come sia invece necessario definire il tipo di grafica (640 x 200), memorizzando il valore 6 nel registro AL.

```
          MOV     AH,00     ;definisce la modalità grafica, cioè
          MOV     AL,06     ;ad alta risoluzione 640 per 200
          INT     10H       ;chiamata di interruzione
          MOV     AL,01     ;punti di colore bianco o nero
          MOV     DX,00     ;posizione iniziale: in alto
          MOV     CX,00     ;a sinistra
ANCORA:   MOV     AH,12     ;parametro per la scrittura di punti
          INT     10H       ;chiamata di interruzione
          INC     DX        ;incremento della posizione verticale
          INC     CX        ;incremento della posizione orizzontale
          INC     CX
          INC     CX
          CMP     DX,0C8H   ;la linea è al limite dello schermo (200)?
          JE      FINE      ;se sì, fine del programma
          JMP     ANCORA    ;se no, si aggiunge alla linea un altro punto
FINE:
```

Questo programma traccia una linea diagonale a partire dall'angolo in alto a sinistra dello schermo (DX = 0 e CX = 0). La linea si compone di punti di colore bianco (AL = 1) e il primo punto viene visualizzato quando il programma entra in ciclo. La posizione verticale (DX) viene incrementata di una unità, mentre la posizione orizzontale (CX) viene incrementata di tre unità. (Poiché la dimensione dello schermo è 640 × 200, la spaziatura orizzontale è in rapporto 3 a 1 con la spaziatura verticale, in modo da visualizzare meglio la linea diagonale). Quando sono stati visualizzati 200 punti in direzione verticale, il programma termina. È possibile eseguire questo programma solo su calcolatori che dispongono di una grafica ad alta risoluzione.

ISTRUZIONI DI MANIPOLAZIONE DI STRINGHE: RICERCA DI UN CARATTERE IN UNA STRINGA

Le istruzioni di manipolazione di stringhe, che vengono utilizzate negli esempi di questo paragrafo, garantiscono una rapida esecuzione dei programmi in linguaggio assemblatore, in quanto permettono di eliminare i cicli di programma, che sono la causa principale del rallentamento dell'esecuzione stessa. Il primo programma ricerca un carattere in una stringa e, in assenza dei comandi di manipolazione di stringhe, avremmo dovuto codificare un ciclo per referenziare ogni carattere della stringa. Il secondo programma copia un'intera stringa da una locazione di memoria ad un'altra, senza ricorrere alla codifica di un ciclo.
Esaminiamo il primo programma:

```
;per macchine IBM 8088/80286
;programma che utilizza un'istruzione di stringa

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
STRINGA   DB      'RICERCA DI UN CARATTERE IN UNA STRINGA'
MSG1      DB      'LA STRINGA CONTIENE LA LETTERA'
MSG2      DB      'LA STRINGA NON CONTIENE LA LETTERA'
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'    ;definisce il segmento di codice
PROCEDURA PROC    FAR              ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS               ;salva DS sullo stack
          SUB     AX,AX            ;azzera AX
          PUSH    AX               ;salva 0 sullo stack
          MOV     AX,DATI          ;indirizzo di DATI in AX
          MOV     DS,AX            ;indirizzo di DATI in DS
          MOV     ES,AX            ;indirizzo di DATI in ES

;routine che esamina la stringa (STRINGA) alla ricerca di un
;carattere
```

```
         CLD                      ;direzione della ricerca: da sinistra a destra
         LEA     DI,STRINGA       ;offset di STRINGA in SI
         MOV     CX,46            ;analisi 46 byte (lettere)
         MOV     AL,'X'           ;ricerca della lettera X in STRINGA
REPNE    SCASB                    ;la ricerca continua finché viene trovata la X
         JCXZ    NESSUNA          ;salta se non è stata trovata alcuna X
         LEA     DX,MSG1          ;X non è in STRINGA
         JMP     STAMPA           ;stampa primo messaggio
NESSUNA: LEA     DX,MSG2          ;X non è in STRINGA, stampa secondo messaggio

;routine che visualizza l'esito della ricerca
STAMPA:
         MOV     AH,09            ;parametro DOS
         INT     21H              ;interruzione DOS

         RET                      ;il controllo ritorna al DOS
PROCEDURA ENDP                    ;fine della procedura
CODICE   ENDS                     ;fine del segmento di codice

         END                      ;fine del programma
```

Il segmento dati contiene tre stringhe: la prima (STRINGA) deve essere analizzata per verificare se contiene un particolare carattere; le altre due (MSG1 e MSG2) specificano l'esito della ricerca.

```
STRINGA  DB  'RICERCA DI UN CARATTERE IN UNA STRINGA'
MSG1     DB  'LA STRINGA CONTIENE LA LETTERA'
MSG2     DB  'LA STRINGA NON CONTIENE LA LETTERA'
```

Quando si utilizzano le istruzioni di stringa, è indispensabile specificare i limiti di stringa entro cui deve essere effettuata la ricerca. In piccoli programmi come questo, tutti i dati sono contenuti in uno stesso segmento e i registri DS e ES contengono entrambi l'indirizzo del segmento. Riportiamo qui di seguito il codice che realizza la ricerca del carattere nella stringa:

```
         CLD                      ;direzione della ricerca: da sinistra a destra
         LEA     DI,STRINGA       ;offset di STRINGA in SI
         MOV     CX,46            ;trasferisce 46 byte (lettere)
         MOV     AL,'X'           ;ricerca della lettera X in STRINGA
REPNE    SCASB                    ;la ricerca continua finché viene trovata la X
         JCXZ    NESSUNA          ;salta se non è stata trovata alcuna X
         LEA     DX,MSG1          ;X non è in STRINGA
         JMP     STAMPA           ;stampa primo messaggio
NESSUNA: LEA     DX,MSG2          ;X non è in STRINGA, stampa
                                  ;secondo messaggio
```

L'istruzione CLD definisce la direzione di scansione della stringa (da sinistra a destra). Il registro DI contiene l'indirizzo della stringa su cui deve essere effettuata la ricerca, mentre il carattere da cercare è contenuto nel registro AL. L'istruzione REPNE SCASB permette di eseguire una ricerca molto rapida, in quanto viene progressivamente scandita la stringa (per un

massimo di 46 passi), fino ad incontrare il carattere indicato. Se questo carattere non è presente nella stringa, il registro CX contiene il valore 0, per cui viene visualizzato sullo schermo un esito negativo (MSG2). Se, invece, la ricerca si è conclusa positivamente, il programma visualizza sullo schermo il messaggio MSG1. Questo tipo di scansione costituisce la base di un analizzatore di parole (si veda il Capitolo 8).

ISTRUZIONI DI MANIPOLAZIONE DI STRINGHE: TRASFERIMENTO DI UNA STRINGA ALL'INTERNO DI UN SEGMENTO

Il seguente programma mostra l'utilizzo di un'altra istruzione di manipolazione di stringhe:

```
;per macchine IBM 8088/80286
;programma che utilizza un'istruzione di stringa

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
STRINGA1  DB      'RICERCA DI UN CARATTERE IN UNA STRINGA'
STRINGA2  DB      46 DUP (?),'$'
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'    ;definisce il segmento di codice
PROCEDURA PROC    FAR              ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS               ;salva DS sullo stack
          SUB     AX,AX            ;azzera AX
          PUSH    AX               ;salva 0 sullo stack
          MOV     AX,DATI          ;indirizzo di DATI in AX
          MOV     DS,AX            ;indirizzo di DATI in DS
          MOV     ES,AX            ;indirizzo di DATI in ES

;routine che trasferisce il contenuto di STRINGA1 in STRINGA2
          LEA     SI,STRINGA1      ;stringa sorgente
          LEA     DI,STRINGA2      ;stringa destinazione
          MOV     CX,46            ;46 byte (lettere) da trasferire
REP       MOVSB                    ;trasferimento

;routine che visualizza la stringa STRINGA2 sullo schermo
          LEA     DX,STRINGA2      ;routine DOS per visualizzazione stringhe
          MOV     AH,09            ;parametro DOS
          INT     21H              ;interruzione DOS

          RET                      ;il controllo ritorna al DOS
PROCEDURA ENDP                     ;fine della procedura
CODICE    ENDS                     ;fine del segmento di codice

          END                      ;fine del programma
```

Questo programma copia semplicemente, e rapidamente, una stringa da una locazione di memoria ad un'altra, utilizzando l'istruzione REP MOVSB, co-

me qui di seguito riportato:

```
        LEA     SI,STRINGA1     ;stringa sorgente
        LEA     DI,STRINGA2     ;stringa destinazione
        MOV     CX,46           ;46 byte (lettere) da trasferire
REP     MOVSB                   ;trasferimento
```

Il registro SI contiene l'indirizzo della stringa sorgente, mentre il registro DI contiene l'indirizzo della stringa destinazione. CX viene inizializzato con il valore 46 e specifica il numero massimo di byte da trasferire quando viene eseguita l'istruzione REP MOVSB. Tutta l'informazione necessaria per eseguire il trasferimento è contenuta in tre linee di programma. Il trasferimento stesso avviene al momento dell'esecuzione dell'istruzione REP MOVSB e risulta quindi estremamente rapido.

6

Direttive all'assemblatore

Le direttive costituiscono un elemento importante della programmazione in linguaggio assemblatore, in quanto informano l'assembler sulla struttura e sulle esigenze del programma.
Più precisamente, una direttiva rappresenta un comando – e non una istruzione da tradurre in codice macchina – che viene interpretato dall'assembler in fase di generazione del codice oggetto e che risulta strettamente dipendente dal tipo di assembler utilizzato.
Nei precedenti capitoli abbiamo sottolineato come il linguaggio assemblatore dipenda in larga misura dal microprocessore che ne esegue le istruzioni. Non a caso la Intel ha deciso di rendere compatibili i suoi microprocessori, in modo che i programmi scritti per il PC IBM (8088) possano essere eseguiti anche sull'AT IBM (80286). Poiché il set di istruzioni dell'80286 è più completo rispetto a quelli disponibili sui microprocessori delle precedenti generazioni, non è garantita la compatibilità in senso inverso.
La Intel ha sentito l'esigenza di rendere compatibili anche gli assembler, per cui l'ASM286 e l'ASM386 sono in grado di tradurre il codice scritto per i microprocessori 8088/80386.
L'Assembler Microsoft e l'Assembler IBM sono prodotti quasi identici (entrambi, infatti, sono stati scritti dalla Microsoft), e hanno dominato negli ultimi anni il mercato per piccoli calcolatori 8088/8086. Quindi, perché un prodotto risulti competitivo, è indispensabile che abbia caratteristiche simili a quelle possedute dai due assembler precedenti. Un altro importante assembler – destinato a piccoli sistemi e disponibile per la famiglia 80286 – è il Turbo Editasm, prodotto dalla Speedware. Questo assembler supporta gran parte delle direttive definite dalla Microsoft, per cui viene garantita una certa compatibilità perfino tra diversi produttori di software.

I programmi presentati nel Capitolo 5 sono stati scritti utilizzando un numero limitato di direttive, perché abbiamo voluto enfatizzare soprattutto le tecniche elementari della programmazione in linguaggio assemblatore. Questo capitolo, invece, si sofferma dettagliatamente sul significato e sulle modalità di utilizzo delle più comuni direttive agli assemblatori IBM, Microsoft e Speedware. Queste direttive saranno utilizzate maggiormente nei programmi presentati nei successivi capitoli del libro, in quanto concorrono a realizzare un tipo di programmazione in linguaggio assemblatore più raffinata e articolata.
Le direttive si possono suddividere in cinque categorie: direttive condizionali, direttive per la definizione di dati, direttive che controllano la stampa dei file, direttive per l'utilizzo delle macro e direttive che specificano la modalità di esecuzione del microprocessore. Esaminiamo in ordine alfabetico le direttive disponibili (a partire dalle direttive numeriche):

.186

Assembler: Microsoft, Speedware

Descrizione: Garantisce la traduzione di istruzioni 80186

Sintassi: .186 (nessun operando)

Commento: Questa direttiva può essere disattivata con la direttiva .8086 e garantisce la traduzione di tutte le istruzioni 8086 e 80186 in modalità non protetta.

Esempio:
```
;il programma viene eseguito su macchine 80186 e si compone di
;istruzioni in linguaggio assemblatore 80186
.186
STACK    SEGMENT PARA STACK
         DB      48 DUP ('STACK ')
STACK    ENDS
```
(Qui di seguito si inserisce il resto del programma)

.286C

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva .286C informa gli assembler di tradurre istruzioni 80286

Sintassi: .286C (nessun operando)

Commento: La modalità di esecuzione 80286 – attivata dalla direttiva .286C – può essere disattivata con la direttiva .8086 e garantisce la traduzione di tutte le istruzioni 8086 e 80286 in modalità non protetta.

Esempio:
```
;il programma utilizza una tabella di lookup per
;calcolare le funzioni logaritmo
.286C
STACK     SEGMENT PARA STACK
          DB        48 DUP ('STACK ')
STACK     ENDS
```
(Qui di seguito si inserisce il resto del programma)

.286P

Assembler: Microsoft

Descrizione: La direttiva .286P è necessaria per tradurre le istruzioni in modalità protetta 80286

Sintassi: .286P (nessun operando)

Commento: La modalità di esecuzione protetta – attivata dalla direttiva .286P – può essere disattivata con la direttiva .8086 e garantisce la traduzione anche di tutte le istruzioni 8086 e 80286 in modalità non protetta.

Esempio:
```
;il programma utilizza una tabella di consultazione per
;calcolare le funzioni logaritmo
.286P
STACK     SEGMENT PARA STACK
          DB        48 DUP ('STACK ')
STACK     ENDS
```
(Qui di seguito si inserisce gran parte del programma)
```
CLTS      ;azzera il flag di task-switch
```
(Qui di seguito si inserisce il resto del programma)

.287

Assembler: Microsoft

Descrizione: La direttiva .287 informa l'assembler Microsoft di tradurre il codice 80287.

Sintassi: .287 (nessun operando)

Commento: La direttiva .287 garantisce che tutte le istruzioni 8087, in aggiunta a quelle che riportiamo qui di seguito, vengano tradotte dall'assembler:

```
FSETPM              ;attiva la modalità protetta
FSTSW    AX         ;memorizza la parola di stato in AX (attesa)
FNSTSW   AX         ;memorizza la parola di stato in AX (no attesa)
```

Esempio:

```
;il programma esegue la sottrazione di alcuni numeri
;modalità di esecuzione 80287
.287
STACK    SEGMENT PARA STACK
         DB      48 DUP ('STACK ')
STACK    ENDS
```

(Qui di seguito si inserisce il resto del programma)

.8086

Assembler: IBM, Microsoft, Speedware

Descrizione: I tre assembler operano in modalità 8088/8086 per difetto. La direttiva .8086 viene utilizzata per disattivare altre modalità di traduzione che sono state attivate precedentemente, come ad esempio .286C.

Sintassi: .8086 (nessun operando)

Commento: Quando gli assembler operano in questa modalità, non eseguono la traduzione di codice 80286.

Esempio:

```
;il programma somma diversi numeri
;modalità di esecuzione 80286
.286C
STACK    SEGMENT PARA STACK
         DB      48 DUP ('STACK ')
STACK    ENDS
```

(Qui di seguito si inserisce la maggior parte del programma)

```
;ripristino della modalità di esecuzione 8086
.8086
```

(Qui di seguito si inserisce il resto del programma)

.8087

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva .8087 informa l'assembler di eseguire la traduzione di istruzioni e dati 8087/80287.

Sintassi: .8087 (nessun operando)

Commento: Questa direttiva ha lo stesso effetto sull'assembler dell'argomento /R nella linea di comando, quando si tratta di assembler IBM o Microsoft.

Esempio:

```
;il programma somma numeri reali
.8087
DATI        SEGMENT PARA 'DATI'
NUMERO1     DQ      7123.45678912345  ;numero reale
NUMERO2     DQ      10.102            ;numero reale
SOMMA       DQ      ?                 ;risultato reale
DATI        ENDS
```

(Qui di seguito si inserisce il resto del programma)

&

Assembler: IBM

Descrizione: La direttiva & (E commerciale o ampersand) viene utilizzata in una espansione di macro per concatenare testo e simboli.

Sintassi: (*testo_o_simbolo*) & (*testo_o_simbolo*)

Commento: Una variabile o un parametro formali – che fanno parte di una stringa o che non seguono un delimitatore – devono essere preceduti dall'operatore & per essere sostituiti nella espansione di macro.

Esempio:

```
NOME        MACRO   PAR         ;macro
REP&PAR     MOV     DX,01234H
```

```
            MOV     AX,'&PAR'
            ENDM
```

(Qui di seguito si inserisce la maggior parte del programma)

```
            NOME    G               ;chiamata alla macro
```

(Viene generato il seguente codice)

```
REPG        MOV     DX,01234H
            MOV     AX,'G'
```

=

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva = (uguale) permette la definizione e la ridefinizione di costanti di programma.

Sintassi: (*etichetta*) = (*valore*)

Commento: L'uso della direttiva = è limitato ad espressioni numeriche. Per il resto, il suo significato è molto simile a quello della direttiva EQU.

Esempio:

```
VAR1  =    1234H   ;può essere ridefinita
VAR2  =    56H     ;può essere ridefinita
VAR1  =    78H     ;può essere ridefinita
VAR3  EQU  3456    ;non può essere ridefinita
```

!

Assembler: IBM

Descrizione: Quando un punto esclamativo (!) precede un carattere, questo può essere utilizzato letteralmente, e perde quindi eventuali significati particolari.

Sintassi: !(*carattere*)

Commento: !(*carattere*) è equivalente a <*carattere*>.

Esempio:

```
!*    !@    !^
```

%

Assembler: IBM

Descrizione: Il simbolo che segue il segno di percentuale (%) viene convertito in un numero espresso nella base numerica selezionata. Nella espansione di macro, questo numero sostituisce la variabile formale.

Sintassi: % (*simbolo*)

Commento: La direttiva % può essere usata solo in una macro

Esempio:

```
NOME   MACRO    V
T      =        V-2
       MOV      AX,%T  ;;risposta in AX
       ENDM
```

%OUT

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva %OUT può essere utilizzata per visualizzare un messaggio alla console, in fase di traduzione del programma. In particolare, la funzione di %OUT è quella di evidenziare lo stato di avanzamento della traduzione del programma.

Sintassi: %OUT (*messaggio*)

Commento: Nessuno

Esempio:

```
%OUT   INIZIA LA TRADUZIONE
```

(Parte del programma si inserisce qui)

```
%OUT   ATTENDERE, LA TRADUZIONE È IN CORSO
```

(Qui di seguito si inserisce il resto del programma)

```
%OUT   TRADUZIONE COMPLETATA
```

;;

Assembler: IBM

Descrizione: Quando due punti e virgola (;;) precedono un commento, il commento stesso non viene incluso nella espansione di macro.

Sintassi: ;; (*stringa_commento*)

Commento: L'uso di due punti e virgola (;;) permette di ridurre lo spazio di memoria occupato quando una macro viene chiamata di frequente.

Esempio:

```
NOME        MACRO   PAR          ;;macro
REP&PAR     MOV     DX,01234H    ;;traferimento
            MOV     AX,'&PAR'
            ENDM                 ;;fine macro
```

ASSUME

Assembler: IBM, Microsoft, Speedware

Descrizione: Questa direttiva informa l'assembler che il segmento deve essere referenziato tramite un registro di segmento. ASSUME NOTHING disattiva le precedenti direttive ASSUME.

Sintassi: ASSUME (*registro_segmento:nome_segmento*)

Commento: CS, DS, ES e SS sono registri di segmento validi

Esempio:

```
CODICE      SEGMENT  PARA 'CODICE' ;definisce il segmento di codice
PROCEDURA   PROC     FAR           ;inizio della procedura
            ASSUME   CS:CODICE,DS:DATI,SS:STACK
            PUSH     DS            ;salva DS sullo stack
            SUB      AX,AX         ;azzera AX
            PUSH     AX            ;salva 0 sullo stack
```

COMMENT

Assembler: IBM, Microsoft, Speedware

Descrizione: Questa direttiva permette di commentare un programma senza ricorrere ai punti e virgola. Viene utilizzata specialmente nel caso di commenti multilinea.

Sintassi: COMMENT (*delimitatore*) *testo* (*delimitatore*)

Commento: I delimitatori da utilizzare sono una scelta del programmatore. Non c'è limite alla lunghezza del commento.

Esempio:
```
COMMENT      @Il testo si trova tra due delimitatori@
```

.CREF / .XCREF
(Cross-REFerence output)

Assembler: IBM, Microsoft, Speedware

Descrizione: .CREF permette la generazione in uscita delle informazioni di cross-reference, quando viene chiamata l'utilità di cross-reference. .XCREF può essere utilizzata per evitare che venga generata in uscita l'informazione di cross-reference.

Sintassi: .CREF (nessun operando)

Commento: .XCREF può utilizzare come operando opzionale un nome simbolico per evitare che venga prodotta in uscita l'informazione di cross-reference relativa al nome citato.

Esempio:
```
DATI       SEGMENT    PARA 'DATI'
INFO       DB         1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
RIS        DB         ?
DATI       ENDS
.XCREF  RIS
CODICE     SEGMENT    PARA 'CODICE' ;definisce il segmento di codice
```
(Qui di seguito si inserisce il resto del programma)

DB
(Define Byte)

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva DB viene utilizzata per definire e inizializzare zone di memoria (multiple) di un byte.

Sintassi: (*variabile*) DB (*espressione*)

Commento: Il nome di variabile è opzionale. L'espressione può essere una costante, una stringa di caratteri oppure una tabella di costanti separate da

una virgola. Se l'espressione è il carattere ? (punto di domanda), viene allocato spazio di memoria non inizializzato, mentre se contiene la direttiva DUP viene allocata in memoria – per un dato numero di volte – l'informazione specificata di seguito. Attraverso questa direttiva è possibile riservare spazio per numeri esadecimali compresi tra 0 e 0FFH o per numeri decimali compresi tra 0 e 255.

Esempio:

```
QUESTO          DB   23
COSTITUISCE     DB   0AH
UN              DB   'G'
ESEMPIO         DB   'POLITECNICO MILANO'
DI              DB   0,1,2,3,4,5,6,7,8,9
UTILIZZO        DB   45 DUP ('STACK')
DELLA           DB   50 DUP (03CH)
DIRETTIVA       DB   'SI',34,0FADH
```

DBIT
(Define BIT)

Assembler: Questa direttiva definisce un particolare tipo di dato per l'80386, riconoscibile dall'assembler ASM386.

Descrizione: La direttiva DBIT viene utilizzata per definire un bit singolo o una stringa di bit. Una stringa può contenere da 1 a 32 cifre binarie e deve terminare con la lettera B.

Sintassi: (*variabile*) DBIT (*espressione*)

Commento: DBIT inizializza un intero byte di memoria

Esempio:

```
ECCO     DBIT   100B         ;byte 00000100
UN       DBIT   1B           ;byte 00000001
ESEMPIO  DBIT   3 DUP (1B)   ;tre byte 00000001
```

DD
(Define Doubleword)

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva DD viene utilizzata per definire e inizializzare zone di memoria (multiple) di 4 byte.

Sintassi: (*variabile*) DD (*espressione*)

Commento: Il nome di variabile è opzionale. L'espressione può essere una costante, una stringa di caratteri oppure una tabella di costanti separate da una virgola. Se l'espressione è il carattere ? (punto di domanda), viene allocato spazio di memoria non inizializzato, mentre se contiene la direttiva DUP viene allocata in memoria – per un dato numero di volte – l'informazione specificata di seguito. Attraverso questa direttiva è possibile riservare spazio per numeri esadecimali compresi tra 0 e 0FFFFFFFFH, per numeri decimali compresi tra 0 e 4294967295 oppure per numeri in notazione scientifica o con virgola decimale (numero memorizzato nel formato in virgola mobile).

Esempio:

```
QUESTO          DD    0FFFFFFFFH
COSTITUISCE     DD    0H
UN              DD    1,22,333,4444,55555,666666,7777777,88888888
ESEMPIO         DD    45H + 23H
DI              DD    2.17654
UTILIZZO        DD    4.567E12
DELLA           DD    50 DUP (03CH)
DIRETTIVA       DD    0.12345E - 2
```

DP

(Define Pointer)

Assembler: Questa direttiva definisce un particolare tipo di dato per l'80386, riconoscibile dall'assembler ASM286.

Descrizione: La direttiva DP viene utilizzata per definire una variabile di 48 bit di tipo PWORD (tipo intero). L'espressione può essere una costante, un nome di segmento, una variabile e una stringa (fino a sei caratteri di lunghezza).

Sintassi: (*variabile*) DP (*espressione*)

Commento: Nessuno

Esempio:

```
AA    DP    'ABCDEF'
BB    DP    ?
CC    DP    3 DUP (0A76543H)
```

DQ
(Define Quadword)

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva DQ viene utilizzata per definire e inizializzare zone di memoria (multiple) di 8 byte.

Sintassi: (*variabile*) DQ (*espressione*)

Commento: Il nome di variabile è opzionale. L'espressione può essere una costante, una stringa di caratteri oppure una tabella di costanti separate da una virgola. Se l'espressione è il carattere? (punto di domanda), viene allocato spazio di memoria non inizializzato, mentre se contiene la direttiva DUP viene allocata in memoria – per un dato numero di volte – l'informazione specificata di seguito. Attraverso questa direttiva è possibile riservare spazio per numeri esadecimali compresi tra 0 e 0FFFFFFFFFFFFFFFFH, per numeri decimali compresi tra 0 e 18446744073709551615 oppure per numeri in notazione scientifica o con virgola decimale (numero memorizzato nel formato in virgola mobile).

Esempio:
```
QUESTO          DQ    100,2000,30000,400000,5000000,60000000
                DQ    700000000,8000000000,90000000000
COSTITUISCE     DQ    1000 DUP (45.678)
UN              DQ    -6.345E-5,+5.0001E45
ESEMPIO         DQ    7989H*12H
```

DT
(Define Tenbyte)

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva DT viene utilizzata per definire e inizializzare zone di memoria (multiple) di 10 byte, destinate a contenere i numeri che vengono utilizzati dai coprocessori 80287/80387.

Sintassi: (*variabile*) DT (*espressione*)

Commento: Il nome di variabile è opzionale. L'espressione può essere una costante, una stringa di caratteri oppure una tabella di costanti separate da una virgola. Se l'espressione è il carattere ? (punto di domanda), viene allocato spazio di memoria non inizializzato, mentre se contiene la direttiva DUP viene allocata in memoria – per un dato numero di volte – l'informazione specificata di seguito. Attraverso questa direttiva è possibile riservare spa-

zio per numeri decimali compattati compresi tra 0 e 9999999999 oppure per numeri in notazione scientifica o con virgola decimale (numero memorizzato nel formato in virgola mobile).

Esempio:

```
QUESTO          DT    100,2000,30000,400000,5000000,60000000
                DT    700000000
COSTITUISCE     DT    1000 DUP (45.678)
UN              DT    -6.345E-5,+5.0001E45
ESEMPIO         DT    7989*12
```

DW

(Define Word)

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva DW viene utilizzata per definire e inizializzare zone di memoria (multiple) di 2 byte.

Sintassi: (*variabile*) DW (*espressione*)

Commento: Il nome di variabile è opzionale. L'espressione può essere una costante, una stringa di caratteri oppure una tabella di costanti separate da una virgola. Se l'espressione è il carattere ? (punto di domanda), viene allocato spazio di memoria non inizializzato, mentre se contiene la direttiva DUP viene allocata in memoria – per un dato numero di volte – l'informazione specificata di seguito. Attraverso questa direttiva è possibile riservare spazio per numeri esadecimali compresi tra 0 e 0FFFFH oppure per numeri decimali compresi tra 0 e 65535.

Esempio:

```
TEST1    DW    1234
TEST2    DW    0ABCDH
TEST3    DW    1,22,333,4444
TEST4    DW    23-12
TEST5    DW    23 DUP (100H)
TEST6    DW    50 DUP (?)
```

ELSE

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva ELSE permette di eseguire una azione alternativa quando viene utilizzata insieme ad un'altra direttiva condizionale.

Sintassi: ELSE (nessun operando)

Commento: Ogni ELSE deve essere preceduta da IFxx. Queste direttive condizionali possono essere annidate e non esiste limitazione al numero di livelli di annidamento.

Esempio:

```
IFNDEF LIB              ;vero se LIB non è definita
INCLUDE B:MACLIB.MAC    ;allora carica questo file
ELSE                    ;altrimenti
INCLUDE B:NEWMAC.LIB    ;carica questo file
ENDIF                   ;fine IFxx
```

END

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva END viene collocata alla fine del programma sorgente. Se END utilizza una espressione opzionale, questa espressione costituisce il puntatore alla prima istruzione eseguibile del programma (entry point).

Sintassi: END (*espressione_opzionale*)

Commento: Se è indispensabile collegare moduli oggetto distinti, solo uno di questi può contenere una espressione opzionale che referenzi il DOS. Se questa espressione è assente, il Linker la identificherà nel primo modulo oggetto.

Esempio:

(Fino a qui si inserisce tutto il programma)

```
            RET                     ;il controllo ritorna al DOS
PROCEDURA   ENDP                    ;fine della procedura
CODICE      ENDS                    ;fine del segmento di codice
            END     PROGRAMMA       ;fine di tutto il programma
```

ENDIF

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva condizionale ENDIF viene utilizzata in coppia con la direttiva condizionale IFxx.

Sintassi: ENDIF (nessun operando)

Commento: Nessuno

Esempio:

```
IFNDEF LIB                ;vero se LIB non è definita
INCLUDE B:MACLIB.MAC      ;allora carica questo file
ELSE                      ;altrimenti
INCLUDE B:NEWMAC.LIB      ;carica questo file
ENDIF                     ;fine IFxx
```

ENDM

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva ENDM delimita inferiormente (termina) la parte di codice delimitata superiormente da una direttiva MACRO, REPT, IRP o IRPC.

Sintassi: ENDM (nessun operando)

Commento: Nessun campo nome è associato con questa direttiva

Esempio:

```
CANCELLA    MACRO       ;cancella lo schermo
MOV         CX,0
MOV         DX,2479H
MOV         BH,7
MOV         AX,0600H
INT         10H
ENDM
```

ENDP

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva ENDP conclude la definizione di una procedura

Sintassi: (*nome_della_procedura*) ENDP

Commenti: È richiesto il campo nome

Esempio:

(Fino a qui si inserisce tutto il programma)

```
                RET                         ;il controllo ritorna al DOS
PROCEDURA       ENDP                        ;fine della procedura
CODICE          ENDS                        ;fine del segmento di codice
                END       PROGRAMMA         ;fine di tutto il programma
```

ENDS

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva ENDS conclude la definizione delle entità SEGMENT e STRUC.

Sintassi: (*nome_del_segmento_o_della_struttura*) ENDS

Commento: È richiesto il nome di un segmento o di una struttura dati

Esempio:

(Fino a qui si inserisce tutto il programma)

```
                RET                         ;il controllo ritorna al DOS
PROCEDURA       ENDP                        ;fine della procedura
CODICE          ENDS                        ;fine del segmento di codice
                END       PROGRAMMA         ;fine di tutto il programma
```

EQU

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva EQU inizializza una costante simbolica ad un certo valore

Sintassi: (*nome_simbolico*) EQU (*valore*)

Commento: La direttiva EQU, diversamente dalla direttiva =, non permette la ridefinizione della costante. EQU non può essere utilizzata con la direttiva STRUC.

Esempio:

```
VALORE1     EQU     1234H      ;fino a 16 bit
VALORE2     EQU     4.56E5     ;virgola mobile
VALORE3     EQU     [SI+6]     ;indice
VALORE4     EQU     DAA        ;istruzione
```

EVEN

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva EVEN aggiorna il program counter al valore pari immediatamente successivo a quello corrente, se questo è dispari.

Sintassi: EVEN (nessun operando)

Commento: Se il program counter è un valore pari, EVEN non lo modifica; in caso contrario, viene inserita una istruzione NOP e il program counter viene aggiornato al valore pari immediatamente successivo.

Esempio:

program counter prima della direttiva EVEN	program counter dopo la direttiva EVEN
0023H	0024H
004AH	004AH

EXITM

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva EXITM permette di uscire da un ciclo o da una macro in base al valore di una certa condizione. EXITM può essere utilizzata con le direttive REPT, IRP, IRPC e MACRO.

Sintassi: EXITM (nessun operando)

Commento: È permesso l'annidamento di blocchi. EXITM permette di uscire solo dal blocco in cui è contenuta.

Esempio:
(Ulteriore codice precede questo punto)

```
IFE      VALORE-34H     ;vero se VALORE=34H
EXITM                   ;esce se è vero
ELSE                    ;altrimenti
INCLUDE  B:NEWMAC.LIB   ;include questo file
ENDIF
ENDM
```

EXTRN

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva EXTRN identifica i simboli o le variabili che vengono utilizzate nel modulo di programma corrente e i cui attributi sono definiti in un altro modulo di programma

Sintassi: EXTRN (*nome* : *tipo*)

Commento: Un simbolo o una variabile vengono dichiarati EXTRN nel modulo di programma in cui sono referenziati e PUBLIC nel modulo di programma in cui sono definiti (supponendo che i due moduli siano distinti). La direttiva EXTRN può essere posta all'interno del segmento di codice o di dati o all'esterno di tutti i segmenti. Il tipo può essere BYTE, WORD, DWORD, QWORD, TBYTE, NEAR, FAR oppure ABS.

Esempio:

```
CODICE      SEGMENT  PARA 'CODICE' ;definisce il segmento di codice
            PUBLIC   VALORE1,VALORE2,RISPOSTA
PROCEDURA   PROC     FAR           ;inizio della procedura
            ASSUME   CS:CODICE,SS:STACK
            PUSH     DS
            SUB      AX,AX
            PUSH     AX
```

(Qui di seguito si inserisce il resto del programma)

(Altro segmento:)

```
            EXTRN    VALORE1,VALORE2,RISPOSTA
NUCODICE    SEGMENT  PARA 'CODICE' ;definisce un altro segmento di
                                    codice
```

(Qui di seguito si inserisce il resto del programma)

GROUP

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva GROUP raggruppa, in un solo segmento di 64 K, tutti i segmenti citati e li associa a uno stesso identificatore simbolico.

Sintassi: (*nome*) GROUP (*nome_del_segmento*)

Commento: L'operando *nome* è l'identificatore sotto cui tutti i segmenti devono essere raggruppati. Questo identificatore deve essere unico tra tutte le etichette di segmento. L'operando *nome_del_segmento* può essere assegnato dalla direttiva SEGMENT oppure da una variabile SEG.

Esempio:

```
                MEGA       GROUP      PICCOLO1,PICCOLO2
                PICCOLO1   SEGMENT    PARA 'CODICE'
                           ASSUME     CS:MEGA
   (Qui di seguito si inserisce tutto il codice per questo segmento)
                PICCOLO1   ENDS

                PICCOLO2   SEGMENT    PARA 'CODICE'
                           ASSUME     CS:MEGA
   (Qui di seguito si inserisce tutto il codice per questo segmento)
                PICCOLO2   ENDS
```

IF [*condizione*]

Assembler: IBM, Microsoft, Speedware

Descrizione: Una direttiva condizionale IF*cond* verifica una certa condizione prima che venga eseguita l'azione seguente. Quando viene utilizzata in coppia con la direttiva ELSE, è possibile specificare anche azioni alternative.

Sintassi:

IF*cond* (*operando*)
(*codice_addizionale*)
ENDIF

Commento: Tutte le direttive condizionali IF*cond* possono essere annidate a qualunque livello. Gli operandi di IF*cond* devono essere riconosciuti alla prima passata dall'assembler.

Direttiva	**Valore**
IF *var1*	;vero se *var1* non è 0
IFE *var1*	;vero se *var1* è 0
IF1 (nessun operando)	;vero se è la prima passata dell'assembler
IF2 (nessun operando)	;vero se è la seconda passata dell'assembler
IFDEF *newval*	;vero se è un simbolo o una variabile definita
IFNDEF *oldval*	;vero se è un simbolo o una variabile indefinita
IFB <*operando*>	;vero se <*operando*> è un carattere spazio
IFNB <*operando*>	;vero se <*operando*> non è un carattere spazio
IFIDN <*operando1*>,<*operando2*>	;vero se <*operando1*> è identico a <*operando2*>
IFDIF <*operando1*>,<*operando2*>	;vero se <*operando1*> non è uguale a <*operando2*>

INCLUDE

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva INCLUDE permette di includere nel modulo corrente il codice sorgente (non tradotto) di un altro file.

Sintassi: INCLUDE (*drive:\percorso\nomefile.est*)

Commento: Questa direttiva è particolarmente utile per includere nel programma un insieme di MACRO (libreria). Il codice sorgente viene caricato immediatamente dopo la direttiva INCLUDE. È possibile avere annidamento di codice, ma – a partire dalla versione DOS 2.0 – la direttiva FILES nel file CONFIG.SYS deve essere inizializzata ad un valore maggiore di quello di default (8).
La versione tradotta del programma evidenzia il codice che è stato incluso, in quanto contiene una C in trentesima colonna. In questo modo viene favorita l'identificazione delle espansioni di codice.

Esempio:
```
IF1
INCLUDE      B:MACLIB.MAC
INCLUDE      C:\INTEL\REGDIS.MAC
ENDIF
```

IRP

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva IRP viene utilizzata per ripetere le azioni che sono comprese tra IRP e ENDM. Il numero di ripetizioni è definito dal numero di operandi presenti nella *lista_di_operandi*. Per ogni ripetizione, l'elemento corrente della *lista_di_operandi* sostituisce ogni occorrenza dell'*operando_formale* sia presente nel blocco di codice.

Sintassi: IRP (*operando_formale*),<*lista_di_operandi*>

Commenti: È richiesta la lista di operandi. Se questa lista è vuota (< >), allora si ripete il blocco di codice una sola volta.

Esempio:
```
IRP     VAL,<2,4,6,8>     ;lista di operandi: 2,4,6,8
DW      VAL               ;i numeri sostituiscono VAL
ENDM
```

IRPC

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva IRPC viene utilizzata per ripetere le azioni comprese tra IRPC e ENDM. Il numero di ripetizioni è stabilito dal numero di caratteri presenti nell'operando *stringa*. In ogni ripetizione, il carattere corrente di *stringa* sostituisce ogni occorrenza dell'*operando_formale* presente nel blocco di codice.

Sintassi: IRPC (*operando_formale*),*stringa*

Commento: Le parentesi angolari per delimitare la stringa sono opzionali

Esempio:

```
IRPC     DATO,2468      ;stringa di caratteri 2, 4, 6, 8
DB       DATO           ;nuovo numero ad ogni ripetizione
DB       DATO*2         ;nuovo numero per 2 ad ogni ripetizione
ENDM                    ;fine del blocco di codice
```

LABEL

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva LABEL permette di definire gli attributi di una etichetta.

Sintassi: (*nome*) LABEL (*tipo*)

Commento: L'operando *nome* può essere un nome qualunque, mentre l'operando *tipo* può essere: BYTE, WORD, DWORD, QWORD oppure TBYTE.

Esempio:

```
TABELLA  LABEL    BYTE
TAB      DW       1234,5678,1111,4545,6778,4598
```

(Qui di seguito si inserisce il codice)

```
         MOV      CL,TABELLA[4]     ;11 in CL
```

.LALL / .SALL / .XALL

Assembler: IBM, Microsoft, Speedware

Descrizione: Queste direttive definiscono il contenuto del file che viene prodotto dalla traduzione.

Sintassi: .LALL (nessun operando)

Commento: .LALL permette di listare le espansioni di tutte le MACRO e dei blocchi condizionali (se viene utilizzata con le direttive .TFCOND, .LFCOND, .SFCOND). Con .SALL viene disattivato il listato di tutte le espansioni di MACRO. .XALL (direttiva di default) permette di listare solo il codice sorgente da cui viene effettivamente generato codice oggetto.

Esempio:

```
            .SALL
CODICE      SEGMENT  PARA 'CODICE' ;definisce il segmento di codice
PROCEDURA   PROC     FAR           ;inizio della procedura
            ASSUME   CS:CODICE,DS:DATI,SS:STACK
            PUSH     DS            ;salva DS sullo stack
            SUB      AX,AX         ;azzera AX
            PUSH     AX            ;salva AX sullo stack
```

.LFCOND

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva .LFCOND permette di listare tutti i blocchi condizionali di codice valutati falsi.

Sintassi: .LFCOND (nessun operando)

Commento: Questa direttiva può essere disattivata con le direttive .SFCOND o .TFCOND.

Esempio:

```
            .LFCOND
CODICE      SEGMENT  PARA 'CODICE' ;definisce il segmento di codice
PROCEDURA   PROC     FAR           ;inizio della procedura
            ASSUME   CS:CODICE,DS:DATI,SS:STACK
            PUSH     DS            ;salva DS sullo stack
            SUB      AX,AX         ;azzera AX
            PUSH     AX            ;salva AX sullo stack
```

.LIST / .XLIST

Assembler: IBM, Microsoft, Speedware

Descrizione: Le direttive .LIST e .XLIST definiscono le parti di codice sorgente che devono essere listate nella versione tradotta del programma.

Sintassi: .LIST (nessun operando)

Commento: .LIST – condizione di default – fa sì che appaia il listato sia del codice sorgente che del codice oggetto, mentre la direttiva .XLIST impedisce questa operazione.

Esempio:

```
            .XLIST
CODICE      SEGMENT   PARA 'CODICE' ;definisce il segmento di codice
PROCEDURA   PROC      FAR           ;inizio della procedura
            ASSUME    CS:CODICE,DS:DATI,SS:STACK
            PUSH      DS            ;salva DS sullo stack
            SUB       AX,AX         ;azzera AX
            PUSH      AX            ;salva AX sullo stack
```

LOCAL

Assembler: IBM, Microsoft, Speedware

Descrizione: Quando viene utilizzata la direttiva LOCAL, un simbolo univoco sostituisce tutte le occorrenze dei corrispondenti nomi simbolici all'interno di una macro, per cui è possibile avere più espansioni della stessa macro in un dato segmento.

Sintassi: LOCAL (*lista_parametri_formali*)

Commenti: La direttiva LOCAL deve essere la prima istruzione in una macro. È lecito l'uso di più istruzioni LOCAL.

Esempio:

```
RITARDO     MACRO                   ;;genera un ritardo software
            LOCAL P1,P2             ;;dichiarazione di P1 e P2 locali
            MOV       DX,15H        ;;circa 5 secondi di ritardo su
                                    ;;PC e XT
P1:         MOV       CX,0FF00H     ;;carica con 65280 (decimale)
P2:         DEC       CX            ;;decrementa di una unità
            JNZ       P2            ;;è zero? se no, salta a P2
            DEC       DX            ;;decrementa DX
            JNZ       P1            ;;è zero? se no, salta a P1
            ENDM
```

MACRO

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva MACRO definisce un segmento di codice che può essere invocato – con parametri differenti – numerose volte in un programma (come una procedura).

Sintassi: (*nome_della_macro*) MACRO (*lista_parametri_formali*)

Commento: Una macro si compone essenzialmente di tre parti: una intestazione, che contiene il nome della macro, la direttiva MACRO e una lista opzionale di parametri formali; un corpo, che contiene il codice della macro; un delimitatore di fine macro, che consiste semplicemente nella direttiva ENDM e che conclude la definizione della macro. Quando le macro vengono espanse nel file di tipo .LST, il loro codice viene preceduto dal simbolo + ad indicare che si tratta di una espansione di macro. La posizione dei parametri formali in lista è fondamentale, in quando stabilisce la corretta corrispondenza con i parametri attuali. Se il numero dei parametri formali è maggiore di quello dei parametri attuali, i parametri in sovrannumero vengono ignorati; se invece si verifica la situazione opposta, ogni elemento non specificato viene sostituito dal carattere null.

Esempio:
```
CANCELLA   MACRO                          ;;pulisce lo schermo, nessuna
                                          ;;variabile
           MOV    CX,0
           MOV    DX,2479H
           MOV    BH,7
           MOV    AX,0600H
           INT    10H
           ENDM

STAMPACHAR MACRO  TESTO                   ;;stampa la stringa passata in
                                          ;;TESTO
           MOV    DX,OFFSET  TESTO        ;;posizione della stringa nel
                                          ;;segmento dati
           MOV    AH,9                    ;;stampa fino alla sentinella '$'
           INT    21H                     ;;stampa dalla posizione corrente
                                          del cursore
           ENDM
```

.MSFLOAT

Assembler: Speedware

Descrizione: Tutti i numeri espressi nel formato in virgola mobile vengono convertiti nel formato numerico Microsoft in virgola mobile.

Sintassi: .MSFLOAT (nessun operando)

Commento: .MSFLOAT può supportare formati in virgola mobile in singola o doppia precisione. I numeri in singola precisione occupano quattro byte: i primi 23 bit costituiscono la mantissa, mentre i restanti 8 contengono l'esponente. I numeri in doppia precisione occupano otto byte: i primi 55 bit costituiscono la mantissa, mentre i restanti 8 bit contengono l'esponente. Questo significa disporre di numeri in singola precisione con 6 o 7 cifre decimali significative e di numeri in doppia precisione con 16 cifre decimali significative.

Esempio:
```
.MSFLOAT
            VALORE1     DD      1.234E-23
            VALORE2     DQ      67.65234987432E35
```

NAME

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva NAME permette di assegnare un nome ad un modulo

Sintassi: NAME (*nome_del_modulo*)

Commento: Ad ogni modulo viene assegnato un nome dall'assembler, rispettando il seguente ordine di preferenze: (1) direttiva NAME, (2) i primi sei caratteri che seguono la direttiva TITLE, oppure (3) i primi sei caratteri del nome del file sorgente.

Esempio:
```
              NAME MODULO
CODICE        SEGMENT    PARA 'CODICE'  ;definisce il segmento di codice
PROCEDURA     PROC       FAR            ;inizio della procedura
              ASSUME     CS:CODICE,DS:DATI,SS:STACK
              PUSH       DS             ;salva DS sullo stack
              SUB        AX,AX          ;azzera AX
              PUSH       AX             ;salva AX sullo stack
```

ORG

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva ORG permette di inizializzare il valore della posizione corrente di inserimento in memoria del codice oggetto (contatore del-

le locazioni di memoria), in modo da definire esplicitamente l'indirizzo di inizio del codice.

Sintassi: ORG (*valore_o_espressione*)

Commento: Il simbolo $ può essere utilizzato come valore corrente del contatore delle locazioni di memoria. L'operando *espressione* deve essere noto alla prima passata dell'assembler e deve definire un numero di 16 bit.

Esempio:
```
ORG     100H
ORG     $
ORG     $ + 100H
```

PAGE

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva PAGE, contenuta nel file sorgente, definisce la lunghezza e l'ampiezza di ogni pagina di stampa.

Sintassi: PAGE (*opzione_1*)(*opzione_2*)

Commento: La direttiva PAGE senza opzioni determina un avanzamento alla pagina successiva. Se invece l'assembler incontra la direttiva PAGE +, incrementa il numero di capitolo. L'operando *opzione1* (numero decimale compreso tra 0 e 255) indica il numero di linee stampate per pagina: il valore di default è 58. L'operando *opzione2* (numero decimale compreso tra 60 e 132) controlla l'ampiezza della pagina: il valore di default è 80. (NOTA: La direttiva PAGE non inizializza la stampante. Questa operazione deve essere eseguita separamente).

Esempio:
```
PAGE          ;avanzamento alla pagina successiva
PAGE ,132     ;ampiezza pagina 132 caratteri
PAGE 20,132   ;lunghezza pagina 20 linee, ampiezza pagina 132
```

PROC

Assembler: IBM, Microsoft, Speedware

Descrizione: PROC identifica un blocco di codice sorgente. Di solito, tutti i programmi contengono almeno una direttiva PROC.

Sintassi: (*nome_della_procedura*) PROC (*tipo*)

Commento: Un blocco di codice identificato dalla direttiva PROC può essere eseguito in linea oppure può essere invocato (se la chiamata viene eseguita da un altro modulo, è necessario definire la procedura come PUBLIC). Se PROC è l'entry point di un file .EXE, oppure se il segmento di codice CS ha un valore diverso da quello in cui è definita la procedura, l'operando *tipo* deve essere FAR. Questo operando può essere NEAR solo se la procedura viene chiamata all'interno dello stesso segmento di codice che la definisce. L'attributo di tipo determina le operazioni che vengono eseguite dal microprocessore in corrispondenza dell'esecuzione dell'istruzione di ritorno RET.

Esempio:

```
CODICE      SEGMENT   PARA 'CODICE'   ;definisce il segmento di codice
PROCEDURA   PROC      FAR             ;inizio della procedura
            ASSUME    CS:CODICE,DS:DATI,SS:STACK
            PUSH      DS              ;salva DS sullo stack
            SUB       AX,AX           ;azzera AX
            PUSH      AX              ;salva AX sullo stack
            CALL      CURSORE         ;chiamata ad una procedura
                                      ;NEAR
```

(Qui di seguito si inserisce altro codice)

```
CURSORE     PROC      NEAR            ;procedura interna allo stesso
                                      ;segmento
            MOV       BX,1            ;inizializza i parametri
            MOV       AH,2
            INT       10H
            RET                       ;estrazione dell'indirizzo di
                                      ;ritorno dallo stack
```

PUBLIC

Assemblatore: IBM, Microsoft, Speedware

Descrizione: La direttiva PUBLIC garantisce che un simbolo definito nel codice sorgente possa essere referenziato da altri programmi. Questa informazione viene interpretata dal Linker in fase di collegamento dei programmi.

Sintassi: PUBLIC (*numero,variabile_o_etichetta*)

Commento: Nessuno

Esempio:

```
CODICE      SEGMENT   PARA 'CODICE'   ;definisce il segmento di codice
PROCEDURA   PROC      FAR             ;inizio della procedura
```

```
ASSUME   CS:CODICE,DS:DATI,SS:STACK
PUSH     DS                ;salva DS sullo stack
SUB      AX,AX             ;azzera AX
PUSH     AX                ;salva AX sullo stack
```

PURGE

Assembler: IBM, Microsoft

Descrizione: La direttiva PURGE cancella la definizione di una macro. Lo spazio precedentemente occupato dalla macro può essere così riutilizzato.

Sintassi: PURGE (*nomi_macro*)

Commento: Una macro non deve necessariamente essere cancellata prima di venire ridefinita.

Esempio:

```
PURGE    CANCELLA
```

.RADIX

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva .RADIX permette di modificare la base del sistema numerico ad un valore compreso tra 2 e 16. Il valore di default è 10.

Sintassi: .RADIX (*valore_specificato_in_un_formato_numerico*)

Commento: .RADIX influisce direttamente sulle variabili definite con le direttive DB e DW, ma non su quelle definite con le direttive DD, DQ e DT. L'uso di un suffisso assicura la non modificabilità della variabile: B (binario), D (decimale), Q o O (ottale), H (esadecimale) e R (esadecimale reale).

Esempio:

```
.RADIX   16
VALORE   DB     120        ;VALORE assume il valore esadecimale 120
.RADIX   2
NUM      DB     10110011   ;NUM assume il valore binario 10110011
.RADIX   10
RIS      DB     120H       ;RIS assume il valore 120H e viene ignorata
                           ;la base 10
```

RECORD

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva RECORD definisce una particolare struttura dati, che si compone di un certo numero di campi contenenti un certo numero di bit, e oltre ad allocarne memoria, ne permette l'accesso.

Sintassi: (*nome_del_record*) RECORD (*nome_del_campo* :*ampiezza*(=esp))

Commento: La direttiva RECORD richiede sia il nome del record che il nome del campo. La dimensione di ogni campo (numero di bit) può variare tra 1 e 16. L'operando 'esp' specifica il valore per difetto della dimensione del campo.

Esempio:
```
ESEMPIO        RECORD  H:1,E:2,L:3,P:6
(cioè:)
HEEL  LLPP  PPPP
(Viene allocata la memoria)
MEM1           MODULE  <1,2,3,4>
(H=1, E=2, L=3 e P=4)
```

REPT

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva REPT garantisce la ripetizione, per un numero specificato di volte, del codice compreso tra REPT e ENDM.

Sintassi: REPT (*valore_o_espressione*)

Commento: Il blocco di codice non deve essere interno ad una MACRO.

Esempio:
```
(Fino a qui è presente altro codice)
VALORE  =       10              ;VALORE inizializzato a 10
        REPT    3               ;tre ripetizioni
        IFE     MEM - VALORE    ;se MEM = VALORE
        EXITM                   ;esce, altrimenti ripete
        ENDIF                   ;fine IF
        ENDM                    ;fine sezione REPT
```

SEGMENT

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva SEGMENT viene utilizzata per delimitare il codice in un programma. Di solito, un programma contiene diversi segmenti di codice.

Sintassi: (*nome_del_segmento*) SEGMENT (*tipo_allineamento*) (*tipo_combinazione*)(*classe*)

Commento: L'operando *tipo_allineamento* può essere PAGE, PARA, WORD oppure BYTE: PAGE definisce l'inizio del segmento in corrispondenza di una pagina di memoria – cioè di un indirizzo divisibile per 256, in cui le due cifre esadecimali meno significative sono pari a 00H; PARA definisce l'inizio del segmento in corrispondenza di un paragrafo di memoria – cioè di un indirizzo divisibile per 16, in cui la cifra esadecimale meno significativa è pari a 0H; WORD definisce l'inizio del segmento in corrispondenza di una word di memoria – cioè di un indirizzo pari, il cui bit meno significativo è uguale a 0H; infine, BYTE definisce l'inizio del segmento in corrispondenza di un byte di memoria – cioè di un indirizzo qualunque.
L'operando *tipo_combinazione* può essere PUBLIC, COMMON, AT(*espressione*), STACK oppure MEMORY (non supportato correntemente): PUBLIC indica che il segmento viene unito ad altri in fase di collegamento; COMMON indica che a questo segmento e ad altri segmenti aventi lo stesso nome viene assegnato lo stesso indirizzo di memoria in fase di collegamento; AT(*espressione*) indica che il segmento inizia dal paragrafo specificato dall'espressione (non viene generato alcun codice per quel segmento); STACK specifica che il segmento costituisce lo stack in fase di esecuzione (è richiesto solo per file .EXE).

Esempio:

```
STACK      SEGMENT  PARA STACK
           DB       48 DUP ('STACK ')
STACK      ENDS
DATI       SEGMENT  PARA 'DATI'
INFO       DB       1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
RIS        DB       ?
DATI       ENDS
CODICE     SEGMENT  PARA 'CODICE' ;definisce il segmento di codice
PROCEDURA  PROC     FAR           ;inizio della procedura
           ASSUME   CS:CODICE,DS:DATI,SS:STACK
           PUSH     DS            ;salva DS sullo stack
           SUB      AX,AX         ;azzera AX
           PUSH     AX            ;salva AX sullo stack
           MOV      AX,DATI       ;preleva i dati in AX
           MOV      DS,AX         ;li mette nel registro DS
```

.SFCOND

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva .SFCOND viene utilizzata per disattivare la stampa di blocchi condizionali di codice che sono valutati False.

Sintassi: .SFCOND (nessun operando)

Commento: .SFCOND non permette la stampa di espressioni condizionali False; .LFCOND disattiva questo stato.

Esempio:

```
            .SFCOND
            IF1
                       INCLUDE B:MACLIB.MAC
            ENDIF
CODICE      SEGMENT    PARA 'CODICE' ;definisce il segmento di codice
            PUBLIC     VALORE1,RIS,SOMMA
PROCEDURA   PROC       FAR           ;inizio della procedura
            ASSUME     CS:CODICE,DS:DATI,SS:STACK
            PUSH       DS            ;salva DS sullo stack
            SUB        AX,AX         ;azzera AX
            PUSH       AX            ;salva AX sullo stack
```

STRUC

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva STRUC è simile alla direttiva RECORD, con l'eccezione che STRUC ha una capacità multibyte.

Sintassi: (*nome_della_struttura*) STRUC

Commento: Questa direttiva permette di allocare memoria per le strutture dati. È possibile ridefinire i contenuti dei campi contenenti un solo elemento (ad esempio una stringa può essere sostituita con un'altra stringa), mentre ciò non è lecito per campi a più elementi.

Esempio:

```
ESEMPIO     STRUC
CAMPO_A     DB      8               ;può essere ridefinito
CAMPO_B     DB      1,2,3           ;non può essere ridefinito
CAMPO_C     DB      'Paolo Rossi'   ;può essere ridefinito
ESEMPIO     ENDS
```

(Allocazione)

```
NOME          ESEMPIO <4,,'Paolo Bianchi'>
```

(Il contenuto del primo e del terzo campo viene sostituito con nuovi valori)

SUBTTL

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva SUBTTL permette la stampa di un sottotitolo sulla linea immediatamente seguente al titolo.

Sintassi: SUBTTL (*stringa*)

Commento: La lunghezza della stringa non deve superare sessanta caratteri, mentre il numero di sottotitoli all'interno di un programma è illimitato. La direttiva SUBTTL senza l'operando *stringa* non determina un salto pagina.

Esempio:

```
            TITLE     FORMATTAZIONE DISCO
            SUBTTL    CONTROLLO CODIFICHE DI ERRORE
            STACK     SEGMENT         PARA STACK
                      DB              48 DUP ('STACK ')
            STACK     ENDS
CODICE      SEGMENT   PARA 'CODICE'   ;definisce il segmento di codice
PROCEDURA   PROC      FAR             ;inizio della procedura
            ASSUME    CS:CODICE,DS:DATI,SS:STACK
            PUSH      DS              ;salva DS sullo stack
            SUB       AX,AX           ;azzera AX
            PUSH      AX              ;salva AX sullo stack
```

.TFCOND

Assembler: IBM, Microsoft

Descrizione: .TFCOND commuta la condizione di controllo delle stampe di blocchi condizionali valutati False. In particolare, conclude le azioni che erano state intraprese con le due direttive .SFCOND e .LFCOND.

Sintassi: .TFCOND (nessun operando)

Commento: .TFCOND commuta rispetto alle condizioni di default

Esempio:

```
.TFCOND
```

TITLE

Assembler: IBM, Microsoft, Speedware

Descrizione: La direttiva TITLE permette la stampa di un messaggio (titolo) sulla seconda linea di ogni pagina.

Sintassi: TITLE (*stringa*)

Commento: In un programma è lecito l'uso di una sola direttiva TITLE. Se questa direttiva è assente, il programma assume lo stesso nome del file sorgente. La lunghezza dell'operando *stringa* non deve superare i 60 caratteri.

Esempio:

```
           TITLE    FORMATTAZIONE DISCO
           SUBTTL   CONTROLLO CODIFICHE DI ERRORE
           STACK    SEGMENT        PARA STACK
                    DB             48 DUP ('STACK ')
           STACK    ENDS
CODICE     SEGMENT  PARA 'CODICE' ;definisce il segmento di codice
PROCEDURA  PROC     FAR            ;inizio della procedura
           ASSUME   CS:CODICE,DS:DATI,SS:STACK
           PUSH     DS             ;salva DS sullo stack
           SUB      AX,AX          ;azzera AX
           PUSH     AX             ;salva AX sullo stack
```

7

Macro, procedure e librerie

Le macro, le procedure e le librerie permettono al programmatore di disporre di codice già corretto e funzionante. In questo modo, lo sviluppo di un programma procede più rapidamente, poiché il programmatore deve progettare e collaudare solo le parti effettivamente nuove, senza dover ogni volta ripartire da zero. Le tecniche di programmazione in linguaggio assemblatore si distinguono a seconda che utilizzino le macro oppure preferiscano ricorrere a procedure o librerie, ma tutte e tre le soluzioni presentano vantaggi e svantaggi.
Questo capitolo si compone di quattro paragrafi, rispettivamente sulle macro, le procedure, le librerie e sui "vantaggi e svantaggi". In quest'ultimo paragrafo, vengono forniti al programmatore dei consigli sulla tecnica di programmazione più adatta – tra quelle discusse nei tre precedenti paragrafi – per codificare la sua applicazione.

7.1 Macro

L'utilizzo di macro costituisce una soluzione versatile nella programmazione in linguaggio assemblatore. Abbiamo già discusso – nel precedente capitolo – la sintassi delle macro, ma ora occupiamoci di indicare la loro funzione.
La direttiva MACRO definisce un insieme di istruzioni in linguaggio assemblatore che possono essere ripetutamente invocate da qualunque parte del programma, referenziando semplicemente il nome della macro. Così facendo, l'assembler ricopia il codice della macro a partire dal punto in cui si è

verificata la chiamata e l'esecuzione del codice stesso avviene in sequenza, cioè senza interrompere il flusso del programma. Ogni macro può essere codificata all'interno del programma che la invoca oppure può fare parte di una libreria (file di macro) il cui accesso viene risolto in fase di traduzione. Una macro – oppure una libreria di macro – contiene infatti codice sorgente che viene tradotto in fase di compilazione, a differenza di quanto accade per una libreria di procedure, che è costituita da parti di programma in codice oggetto, collegate mediante il linker.
Gli assembler si sono evoluti di pari passo con i microprocessori Intel 8088/80386. I primi assembler infatti non erano in grado di interpretare le istruzioni dell'8087/80387 e neppure di supportare la modalità di indirizzamento virtuale protetta tipica dell'80286/80386. In quest'ottica, le macro potevano essere utilizzate per sopperire alle carenze dell'assembler, per quanto riguarda la codifica di istruzioni che non sono state definite a livello di microprogramma. Ad esempio, se si desidera realizzare una operazione di incremento dello stack-pointer per l'8087 e si dispone di un assembler che non supporta l'8087, è possibile risolvere il problema codificando la seguente macro:

```
INCSP   MACRO          ;;incrementa lo stack – pointer
        WAIT           ;;sincronizzazione tra 8087 e 8088
        ESC 0EH,DI     ;;invio della sequenza di codice corretta
        ENDM           ;;fine della macro
```

Una macro di questo tipo può sostituire ogni istruzione dell'8087. La tecnica programmativa appena descritta è sicuramente dispendiosa in termini di tempo di sviluppo quando il numero delle istruzioni da codificare risulta elevato.
Se si dispone della versione 2.0 del Macro Assembler IBM oppure della versione 3.0 dell'Assembler Microsoft, non vengono supportate le istruzioni di controllo protette dell'80286/80386, ma devono essere simulate con l'aiuto delle macro. Al contrario, la versione 1.02 dell'Assembler Speedware è in grado di supportare tutte le 26 istruzioni di controllo protette.
La funzione delle macro è soprattutto quella di definire parti di codice che possono essere invocate più volte all'interno di un programma e che realizzano alcune operazioni come la cancellazione dello schermo, il controllo dei movimenti del cursore oppure operazioni aritmetiche.

STRUTTURA DI UNA MACRO

Nel precedente capitolo abbiamo trattato la sintassi di una macro. Ricordiamo che una macro si compone di tre parti fondamentali: una intestazione, un corpo e un delimitatore di fine macro (ENDM). L'intestazione contiene il nome della macro, la direttiva MACRO e, facoltativamente, un certo nu-

```
PAGE ,132
;per macchine IBM 8088/80286 con monitor RGB
;il programma illustra l'uso di una semplice macro

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
BACK      DB      2000 DUP (' ')
DATI      ENDS

RITARDO   MACRO   TEMPO
          LOCAL   P1,P2        ;;P1 e P2 sono etichette locali
          PUSH    DX           ;;salva i valori originali di DX
          PUSH    CX           ;;e di CX
          MOV     DX,TEMPO     ;;argomento TEMPO in DX
P1:       MOV     CX,0FF00H    ;;carica in CX il contatore 00FF00H
P2:       DEC     CX           ;;decrementa CX: il tempo passa
          JNZ     P2           ;;se diverso da zero, continua
          DEC     DX           ;;se CX = 0, decrementa DX
          JNZ     P1           ;;se DX è diverso da zero, carica CX di nuovo
          POP     CX           ;;se DX = 0, ripristina i valori di CX
          POP     DX           ;;e di DX
          ENDM                 ;;fine della macro

CODICE    SEGMENT PARA 'CODICE'  ;definisce il segmento di codice
PROCEDURA PROC    FAR          ;inizio della procedura
          ASSUME  CS:CODICE,ES:DATI,SS:STACK
          PUSH    DS           ;salva DS sullo stack
          SUB     AX,AX        ;azzera AX
          PUSH    AX           ;salva 0 sullo stack
          MOV     AX,DATI      ;indirizzo di DATI in AX
          MOV     ES,AX        ;indirizzo di DATI in ES

;questo segmento di programma esegue la cancellazione dello
;schermo, tramite visualizzazione di 80 per 25 caratteri spazio.
;Ripetendo la stessa operazione con un valore differente nel
;registro BL, il colore di tutto lo schermo viene modificato! La
;macro RITARDO garantisce per un periodo di tempo specificato
;l'immutabilità del colore sullo schermo.
          MOV     CX,08H       ;ripete il ciclo 8 volte
          MOV     BL,00H       ;colore originale per lo sfondo
ANCORA:   LEA     BP,BACK      ;stringa di caratteri spazio
          MOV     DX,0000      ;cursore in alto a sinistra
          MOV     AH,19        ;attributo di stringa
          MOV     AL,1         ;stampa dei caratteri e aggiornamamento cursore
          PUSH    CX           ;salva il contatore
          MOV     CX,07D0H     ;visualizzazione di 2000 caratteri spazio
          INT     10H          ;chiamata di interruzione
          RITARDO 10           ;attesa di 10 unità di tempo
          ADD     BL,10H       ;modifica del colore dello sfondo
          POP     CX           ;ripristina il contatore originale
          LOOP    ANCORA       ;operazione ripetuta 8 volte

          RET                  ;il controllo ritorna al DOS
PROCEDURA ENDP                 ;fine della procedura
CODICE    ENDS                 ;fine del segmento di codice

          END                  ;fine del programma
```

Figura 7.1 Programma che evidenzia l'uso di una macro

mero di argomenti formali, utilizzati per adattare la macro al contesto di utilizzo. Il corpo della macro contiene il codice che viene espanso in ogni punto del programma in cui la macro è invocata, mentre il delimitatore di fine macro è la direttiva ENDM. Per meglio comprendere l'utilizzo di una macro, esaminiamo il programma di Figura 7.1, che presenta una struttura analoga a quella dei programmi discussi nel Capitolo 5, ma che si differenzia per la presenza della macro RITARDO.
In particolare, riportiamo qui di seguito la parte di programma contenente la macro:

```
RITARDO  MACRO   TEMPO
         LOCAL   P1,P2          ;;P1 e P2 sono etichette locali
         PUSH    DX             ;;salva i valori originali di DX
         PUSH    CX             ;;e di CX
         MOV     DX,TEMPO       ;;argomento TEMPO in DX
P1:      MOV     CX,0FF00H      ;;carica in CX il contatore 00FF00H
P2:      DEC     CX             ;;decrementa CX: il tempo passa
         JNZ     P2             ;;se diverso da zero, continua
         DEC     DX             ;;se CX = 0, decrementa DX
         JNZ     P1             ;;se DX è diverso da zero, carica CX
                                ;;di nuovo
         POP     CX             ;;se DX = 0, ripristina i valori di CX
         POP     DX             ;;e di DX
         ENDM                   ;;fine della macro
```

Il nome della macro è RITARDO. Quando, nel corso del programma, l'assembler incontra l'identificatore simbolico RITARDO, ricopia, a partire da questa locazione di memoria, il corpo della macro che è stata invocata. Conclusa questa operazione, l'argomento TEMPO viene trasferito nel registro DX. La direttiva LOCAL definisce due etichette (P1 e P2) locali alla macro, per evitare che la stessa etichetta venga referenziata in più espansioni di macro. Nel nostro esempio, infatti, ad ogni invocazione di macro, viene definita una coppia distinta di etichette. La funzione della macro RITARDO è quella di simulare un'attesa a tempo durante l'esecuzione del programma, decrementando il contenuto dei registri DX e CX. DX memorizza il numero di attese, mentre CX contiene il valore di ogni attesa. Chiaramente i ritardi hardware sono più accurati, ma la funzione di questa macro è esclusivamente di carattere didattico.
Nel Capitolo 5, abbiamo presentato un programma che permette di modificare il colore dello schermo e di visualizzare una immagine su di esso.
Durante l'esecuzione del programma di Figura 7.1, il colore dello schermo si modifica automaticamente per otto volte. Questo risultato si raggiunge sommando il valore numerico 10H al contenuto corrente del registro BL, ogni volta che il programma entra in ciclo. La macro RITARDO controlla la durata di ogni colorazione dello schermo e il valore che assume il suo argomento formale è stato determinato sperimentalmente per il calcolatore AT

IBM funzionante ad una frequenza di 9 MHz. (Se il calcolatore a vostra disposizione esegue ad una velocità differente, aggiornate opportunamente questo valore). La Figura 7.2 mostra il contenuto del file .LST, in cui il codice della macro è stato espanso immediatamente dopo l'istruzione di chiamata della macro.
Si noti in particolare come il codice originale della macro (riconoscibile dai commenti preceduti da due punti e virgola), non presenti alla sua sinistra

```
                        PAGE ,132
                        ;per macchine IBM 8088/80286 con monitor RGB
                        ;il programma illustra l'uso di una semplice macro

0000                    STACK    SEGMENT PARA STACK
0000   40 [                      DB      64 DUP ('MYSTACK ')
            4D 59 53 54
            41 43 4B 20
                       ]
0020                    STACK    ENDS

0000                    DATI     SEGMENT PARA 'DATI'
0000 07D0 [             BACK     DB      2000 DUP (' ')
             20

07D0                    DATI     ENDS

                        RITARDO  MACRO   TEMPO
                                 LOCAL   P1,P2        ;;P1 e P2 sono etichette locali
                                 PUSH    DX           ;;salva i valori originali di DX
                                 PUSH    CX           ;;e di CX
                                 MOV     DX,TEMPO     ;;argomento TEMPO in DX
                        P1:      MOV     CX,0FF00H    ;;carica in CX il contatore 00FF00H
                        P2:      DEC     CX           ;;decrementa CX: il tempo passa
                                 JNZ     P2           ;;se diverso da zero, continua
                                 DEC     DX           ;;se CX = 0, decrementa DX
                                 JNZ     P1           ;;se DX è diverso da zero, carica CX di nuovo
                                 POP     CX           ;;se DX = 0, ripristina i valori di CX
                                 POP     DX           ;;e di DX
                                 ENDM                 ;;fine della macro

0000                    CODICE   SEGMENT PARA 'CODICE' ;definisce il segmento di codice
0000                    PROCEDURA PROC   FAR          ;inizio della procedura
                                 ASSUME  CS:CODICE,ES:DATI,SS:STACK
0000 1E                          PUSH    DS           ;salva DS sullo stack
0001 2B C0                       SUB     AX,AX        ;azzera AX
0003 50                          PUSH    AX           ;salva 0 sullo stack
0004 B8 ----R                    MOV     AX,DATI      ;indirizzo di DATI in AX
0007 8E C0                       MOV     ES,AX        ;indirizzo di DATI in ES

                        ;questo segmento di programma esegue la cancellazione dello
                        ;schermo, tramite visualizzazione di 80 per 25 caratteri
                        ;spazio.
                        ;Ripetendo la stessa operazione con un valore differente nel
                        ;registro BL, il colore di tutto lo schermo viene modificato! La
                        ;macro RITARDO garantisce per un periodo di tempo specificato
                        ;l'immutabilità del colore sullo schermo.
0009 B9 0008                     MOV     CX,08H       ;ripete il ciclo 8 volte
000C B3 00                       MOV     BL,00H       ;colore originale per lo sfondo
```

```
000E 8D 2E 0000 R      ANCORA:    LEA       BP,BACK        ;stringa di caratteri spazio
0012 BA 0000                      MOV       DX,0000        ;cursore in alto a sinistra
0015 B4 13                        MOV       AH,19          ;attributo di stringa
0017 B0 01                        MOV       AL,1           ;stampa dei caratteri e aggiornamamento cursore
0019 51                           PUSH      CX             ;salva il contatore
001A B9 07D0                      MOV       CX,07D0H       ;visualizzazione di 2000 caratteri spazio
001D CD 10                        INT       10H            ;chiamata di interruzione
                                  RITARDO 10               ;attesa di 10 unità di tempo
001F 52             +             PUSH      DX
0020 51             +             PUSH      CX
0021 BA 000A        +             MOV       DX,10
0024 B9 FF00        +  ??0000:    MOV       CX,0FF00H
0027 49             +  ??0001:    DEC       CX
0028 75 FD          +             JNZ       ??0001
002A 4A             +             DEC       DX
002B 75 F7          +             JNZ       ??0000
002D 59             +             POP       CX
002E 5A             +             POP       DX
002F 80 C3 10                     ADD       BL,10H         ;modifica del colore dello sfondo
0032 59                           POP       CX             ;ripristina il contatore originale
0033 E2 D9                        LOOP      ANCORA         ;operazione ripetuta 8 volte

                                  RET                      ;il controllo ritorna al DOS
0035 CB                PROCEDURA ENDP                      ;fine della procedura
0036                   CODICE     ENDS                     ;fine del segmento di codice

0036                              END                      ;fine del programma

Macros:

            Name                  Length

RITARDO.................          0007

Segments and Group:

            Name                  Size   Align   Combine   Class

CODICE..................          0036   PARA    NONE      'CODICE'
DATI....................          07D0   PARA    NONE      'DATI'
STACK...................          0200   PARA    STACK

Symbols:

            Name                  Type   Value   Attr

ANCORA..................          L NEAR 000E    CODICE
BACK....................          L BYTE 0000    DATI     Length =07D0
PROCEDURA...............          F PROC 0000    CODICE   Length =0036
??0000..................          L NEAR 0024    CODICE
??0001..................          L NEAR 0027    CODICE

49928 Bytes free

Warning Severe
Errors  Errors
0       0
```

Figura 7.2 File .LST, contenente la versione tradotta del programma di Figura 7.1, che evidenzia l'espansione della macro

la corrispondente traduzione in codice macchina, come invece accade per l'espansione di macro generata dall'assembler dopo l'istruzione RITARDO 10:

```
                                  RITARDO    10
001F   52           +             PUSH       DX
0020   51           +             PUSH       CX
0021   BA  000A     +             MOV        DX,10
0024   B9  FF00     +  ??0000:    MOV        CX,0FF00H
0027   49           +  ??0001:    DEC        CX
0028   75  FD       +             JNZ        ??0001
002A   4A           +             DEC        DX
002B   75  F7       +             JNZ        ??0000
002D   59           +             POP        CX
002E   5A           +             POP        DX
```

Il simbolo + identifica l'espansione di macro e questa notazione risulta molto utile in fase di correzione del programma.
Poiché, nel nostro caso, la macro RITARDO è stata espansa una sola volta, la dichiarazione LOCAL non è strettamente necessaria. Si tenga comunque presente che – con questa dichiarazione – P1 e P2 vengono sostituiti con le due espressioni ??0000 e ??0001. Se la macro RITARDO fosse stata invocata una seconda volta all'interno dello stesso programma, P1 e P2 avrebbero assunto le espressioni ??0002 e ??0003. La definizione di etichette locali ad una macro evita quindi la possibilità che si verifichino errori di programmazione.

LIBRERIA DI MACRO

Le macro possono essere invocate solo dal programma in cui sono definite – come è stato indicato in Figura 7.2 – oppure possono fare parte di una libreria di macro e quindi possono essere invocate da qualunque programma. La soluzione di collocare in una libreria un certo numero di macro di utilizzo frequente (che il programmatore può invocare tramite una semplice citazione, evitando così di dover codificare ogni macro esplicitamente all'interno del programma stesso) ottimizza il tempo si sviluppo del programma. La Figura 7.3 mostra un insieme di macro che appartengono alla libreria MACLIB.MAC. Si tenga presente che una libreria di macro contiene solo codice sorgente, per cui l'unica operazione da effettuare è quella di salvare il file ASCII sul dischetto, al termine della scrittura delle macro.
PUSHA e POPA sono due macro che possono essere invocate solo dai programmi che vengono eseguiti su calcolatori dotati di microprocessori 8088 e 8086, in quanto i microprocessori 80286/80386 supportano due istruzioni (PUSHA e POPA) che realizzano le stesse funzioni a livello di microprogramma. Se state utilizzando uno tra questi due ultimi microprocessori, specifi-

```
PUSHA     MACRO               ;;salva i registri macchina
          PUSH   AX           ;;non è necessaria su 80286/80386
          PUSH   CX
          PUSH   DX
          PUSH   BX
          PUSH   SP
          PUSH   BP
          PUSH   SI
          PUSH   DI
          ENDM

POPA      MACRO
          POP    DI           ;;ripristina i registri macchina
          POP    SI           ;;non è necessaria su 80286/80386
          POP    BP
          POP    SP
          POP    BX
          POP    DX
          POP    CX
          POP    AX
          ENDM

RITARDO   MACRO  TEMPO        ;;attesa a tempo controllata software
          LOCAL  P1,P2        ;;dall'argomento TEMPO e dal clock di sistema
          PUSHA               ;;salva i registri
          MOV    DX,TEMPO     ;;numero di attese in DX
P1:       MOV    CX,0FF00H    ;;tempo di ogni attesa in CX
          DEC    CX           ;;decrementa CX
          JNZ    P2           ;;se non è zero, P2
          DEC    DX           ;;se è zero, decrementa DX
          JNZ    P1           ;;se DX non è zero, P1
          POPA                ;;ripristino dei registri
          ENDM                ;;fine della macro

CANCELLA  MACRO               ;;cancella lo schermo
          PUSHA               ;;salva i registri
          MOV    CX,0         ;;angolo superiore della finestra
          MOV    DX,2476H     ;;angolo inferiore della finestra
          MOV    BH,7         ;;attributo di schermo (normale)
          MOV    AX,0600H     ;;parametri di interruzione
          INT    10H          ;;chiamata di interruzione
          POPA                ;;ripristino dei registri
          ENDM                ;;fine della macro

STAMPANUM MACRO               ;;visualizza un numero ASCII sullo schermo
          PUSHA               ;;salva i registri
          PUSH   AX           ;;salva AX perchè sarà modificato
          MOV    AH,15        ;;stato di visualizzazione
          INT    10H          ;;chiamata di interruzione, uscita in BX
          POP    AX           ;;ripristina il valore di AX
          AND    AL,0FH       ;;azzera i quattro bit più significativi
          OR     AL,30H       ;;conversione ASCII
          MOV    AH,10        ;;parametro di interruzione
          MOV    CX,1         ;;numero di caratteri da visualizzare
          INT    10H          ;;chiamata di interruzione
          POPA                ;;ripristino dei registri
          ENDM                ;;fine della macro

CURSORE   MACRO  LOC          ;;posiziona il cursore in LOC
          PUSHA               ;;salva i registri
```

```
            MOV    AH,15       ;;stato di visualizzazione
            INT    10H         ;;chiamata di interruzione, uscita in BX
            MOV    DX,LOC      ;;posizione sullo schermo in DX
            MOV    AH,2        ;;parametro relativo al cursore
            INT    10H         ;;chiamata di interruzione
            POPA               ;;ripristino dei registri
            ENDM               ;;fine della macro

STAMPACHAR MACRO  TESTO       ;;visualizza una stringa di caratteri.
            PUSHA              ;;La stringa viene passata alla macro
            LEA    DX,TESTO    ;;attraverso l'argomento formale TESTO.
            MOV    AH,9        ;;La visualizzazione inizia dalla posizione
            INT    21H         ;;corrente del cursore e procede fino a
            POPA               ;;quando si incontra il tappo '$'.
            ENDM               ;;fine della macro
```

Figura 7.3 Insieme di macro di utilizzo frequente contenute in un file denominato MACLIB.MAC

cate, all'inizio del programma, le direttive 0.286C oppure 0.386C, per essere sicuri che gli assembler interpretino correttamente le istruzioni PUSHA e POPA.

Nella libreria MACLIB.MAC, le macro PUSHA e POPA eseguono il salvataggio e il ripristino dello stato corrente dei registri della macchina, per preservarli da modifiche indesiderate, mentre la macro RITARDO coincide con quella di Figura 7.1; la macro CANCELLA è stata già discussa nel Capitolo 5, ma ora, in quanto appartenente ad una libreria, può essere invocata più volte, senza che il suo codice appaia esplicitamente nel programma, mentre la macro STAMPANUM permette di visualizzare sullo schermo un numero a singola cifra espresso nel formato ASCII. Si ricordi, a questo proposito, che il numero esadecimale 05H, contenuto nel registro AL, deve essere prima convertito nella codifica ASCII perché possa essere visualizzato sullo schermo; la macro CURSORE utilizza il valore dell'argomento formale LOC per definire le posizioni verticale (DH) e orizzontale (DL) occupate dal cursore su uno schermo di dimensione pari a 80 × 25, mentre la macro STAMPACHAR esegue la visualizzazione di una stringa di caratteri ASCII a partire dalla posizione corrente del cursore. Questa operazione continua fino a quando la macro incontra il carattere tappo $ che delimita la fine della stringa.

Per rendersi conto di come una libreria di macro possa facilitare lo sviluppo di un programma, esaminiamo la Figura 7.4, in cui viene mostrato un programma che utilizza alcune macro. Questo programma cancella lo schermo, visualizza – in cima allo schermo – il messaggio 'Semplice programma di conteggio' e – al centro dello schermo – il conteggio da 0 a 9 (il programma visualizza in realtà solo la cifra meno significativa del vero contatore contenuto nel registro AL: quindi può essere visualizzato un 5 sullo schermo anche se in AL il contatore vale, ad esempio, 45). L'esecuzione termina dopo cinque cicli.

```
;per macchine IBM 8088/80286
;programma che illustra l'uso di una libreria di macro

IF1                             ;direttiva per caricare la
        INCLUDE B:MACLIB.MAC    ;libreria di macro dal drive B
ENDIF

STACK   SEGMENT PARA STACK
        DB      48 DUP ('STACK ')
STACK   ENDS

DATI    SEGMENT PARA 'DATI'
MESSAGGIO DB    'Semplice programma di conteggio'
DATI    ENDS

CODICE  SEGMENT PARA 'CODICE'   ;definisce il segmento di codice
PROCEDURA PROC  FAR             ;inizio della procedura
        ASSUME  CS:CODICE,DS:DATI,SS:STACK
        PUSH    DS              ;salva DS sullo stack
        SUB     AX,AX           ;azzera AX
        PUSH    AX              ;salva 0 sullo stack
        MOV     AX,DATI         ;indirizzo di DATI in AX
        MOV     DS,AX           ;indirizzo di DATI in DS

        CANCELLA                ;chiamata della macro CANCELLA
        CURSORE 0019H           ;messaggio al centro dello schermo
        STAMPACHAR MESSAGGIO    ;visualizzazione del messaggio

        MOV     AX,00           ;contatore inizializzato a zero
ANCORA: CURSORE 0C28H           ;posizionamento al centro dello schermo
        STAMPANUM               ;visualizzazione del contenuto di AL
        RITARDO 10              ;attesa di 10 istanti di tempo
        ADD     AL,01           ;somma di una unità al registro AL
        DAA                     ;sistemazione decimale del risultato
        CMP     AL,50H          ;dopo cinque cicli, fine
        JE      FINE
        JMP     ANCORA          ;ciclo successivo
FINE:   CANCELLA                ;ulteriore cancellazione dello schermo

        RET                     ;il controllo ritorna al DOS
PROCEDURA ENDP                  ;fine della procedura
CODICE  ENDS                    ;fine del segmento di codice

        END                     ;fine del programma
```

Figura 7.4 Programma di conteggio che evidenzia il vantaggio di disporre di una libreria di macro

Per includere nel programma la libreria di macro abbiamo utilizzato la direttiva IF1 e, per specificare il drive in cui risiede la libreria e il nome della libreria, ci siamo serviti della direttiva INCLUDE.

```
IF1                                 ;direttiva per caricare la
          INCLUDE B:MACLIB.MAC      ;libreria di macro dal drive B
ENDIF
```

È evidente come queste tre linee di codice evitino al programmatore una codifica ben più estesa di istruzioni.

```
CANCELLA                  ;chiamata della macro CANCELLA
CURSORE 0019H             ;messaggio al centro dello schermo
STAMPACHAR MESSAGGIO      ;visualizzazione del messaggio
```

La macro CANCELLA cancella lo schermo, mentre la macro CURSORE posiziona il cursore sulla prima linea dello schermo e in colonna 25 (19H), in modo tale che il messaggio risulti centrato; la macro STAMPACHAR esegue la visualizzazione della stringa MESSAGGIO.
Le seguenti istruzioni permettono di visualizzare sullo schermo il conteggio da 0 a 9.

```
          MOV         AX,00     ;contatore inizializzato a zero
ANCORA:   CURSORE     0C28H     ;posizionamento al centro
                                ;dello schermo
          STAMPANUM             ;visualizzazione del contenuto di AL
          RITARDO     10        ;attesa di 10 istanti di tempo
          ADD         AL,01     ;somma di una unità al registro AL
          DAA                   ;sistemazione decimale
                                ;del risultato
          CMP         AL,50H    ;dopo cinque cicli, fine
          JE          FINE
          JMP         ANCORA    ;ciclo successivo
FINE:     CANCELLA              ;ulteriore cancellazione
                                ;dello schermo
```

Quando viene chiamata la macro STAMPANUM, la cifra contenuta nei quattro bit meno significativi del registro AL viene convertita in un carattere ASCII e visualizzata sullo schermo. Il conteggio decimale progressivo viene garantito dalle due istruzioni ADD e DAA. Prima che il programma termini, lo schermo viene cancellato di nuovo.
È interessante tradurre il programma ed esaminare il contenuto del file .LST prodotto dall'assembler, per rendersi conto della grande quantità di codice che viene aggiunto a quello effettivamente codificato dal programmatore.

7.2 Procedure

Tutti i programmi che abbiamo discusso finora hanno utilizzato una sola procedura. Un programma, invece, può contenere più di una procedura, che può essere di tipo NEAR, se viene definita nello stesso segmento che la invoca, oppure FAR, se viene definita in un segmento distinto da quello che la chiama. Una procedura si compone di una sequenza di istruzioni che vengo-

no eseguite su richiesta del programmatore in un punto qualsiasi del programma. Nel prossimo paragrafo, esamineremo entrambi questi tipi di procedure.

STRUTTURA DI UNA PROCEDURA

Un programma nel linguaggio assemblatore si compone essenzialmente di un segmento di codice, un segmento di dati e un segmento di stack. In particolare, nel prossimo esempio, il nome simbolico CODICE identifica il segmento, in cui è allocato il codice del programma, che può contenere la definizione di diverse procedure.

```
CODICE      SEGMENT  PARA 'CODICE'   ;definisce il segmento di codice
PROCEDURA   PROC     FAR             ;inizio della procedura
            ASSUME   CS:CODICE,DS:DATI,SS:STACK
            PUSH     DS              ;salva DS sullo stack
            SUB      AX,AX           ;azzera AX
            PUSH     AX              ;salva 0 sullo stack
            MOV      AX,DATI         ;indirizzo di DATI in AX
            MOV      DS,AX           ;indirizzo di DATI in DS
```

(Qui si inserisce il corpo della procedura)

```
            RET                      ;il controllo ritorna al DOS
PROCEDURA   ENDP                     ;fine della procedura
CODICE      ENDS                     ;fine del segmento di codice
            END                      ;fine del programma
```

Il nome della procedura è PROCEDURA, mentre il suo tipo è FAR ad indicare che viene invocata da un segmento di codice distinto da CODICE. Tutte le procedure presentano una struttura simile, ma si differenziano a seconda del tipo (NEAR o FAR). Si tenga presente che, prima di terminare una procedura (PROCEDURA ENDP), è indispensabile codificare l'istruzione RET.
Gli attributi di tipo NEAR e FAR definiscono le modalità di chiamata (CALL) alla procedura e di ritorno al programma chiamante, dopo che la procedura è stata eseguita. Infatti, se la procedura è di tipo NEAR, il microprocessore salva sullo stack solo il contenuto del registro IP (program counter), mentre se la procedura è di tipo FAR vengono salvati sullo stack sia il contenuto del registro IP che il contenuto del registro CS (segmento codice).
La Figura 7.5 mostra un programma che risulta molto simile a quello presentato all'inizio del capitolo. L'unica differenza esistente tra i due programmi è che ora la routine RITARDO non è strutturata più come macro, ma come procedura di tipo NEAR. Esaminate attentamente il codice della Figura 7.1 per rendervi conto delle modifiche esistenti tra i due programmi. Nella parte di programma qui di seguito riportata l'istruzione di invocazione della

```
PAGE ,132
;per macchine IBM 8088/80286 con monitor RGB
;il programma illustra l'uso di una procedura NEAR

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
BACK      DB      2000 DUP (' ')
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE' ;definisce il segmento di codice
PROCEDURA PROC    FAR           ;inizio della procedura
          ASSUME  CS:CODICE,ES:DATI,SS:STACK
          PUSH    DS            ;salva DS sullo stack
          SUB     AX,AX         ;azzera AX
          PUSH    AX            ;salva 0 sullo stack
          MOV     AX,DATI       ;indirizzo di DATI in AX
          MOV     ES,AX         ;indirizzo di DATI in ES

;questo segmento di programma cancella lo schermo, visualizzando
;80 per 25 caratteri spazio.
;Ripetendo la stessa operazione con un valore differente nel
;registro BL, il colore di tutto lo schermo viene modificato! La
;procedura RITARDO garantisce per un periodo di tempo specificato
;l'immutabilità del colore sullo schermo.
          MOV     CX,08H        ;ripete il ciclo 8 volte
          MOV     BL,00H        ;colore originale per lo sfondo
ANCORA:   LEA     BP,BACK       ;stringa di caratteri spazio
          MOV     DX,0000       ;cursore in alto a sinistra
          MOV     AH,19         ;attributo di stringa
          MOV     AL,1          ;stampa dei caratteri e aggiornamento cursore
          PUSH    CX            ;salva il contatore
          MOV     CX,07D0H      ;visualizzazione di 2000 caratteri spazio
          INT     10H           ;chiamata di interruzione
          CALL    RITARDO       ;chiamata della procedura NEAR RITARDO
          ADD     BL,10H        ;modifica del colore dello sfondo
          POP     CX            ;ripristina il contatore originale
          LOOP    ANCORA        ;operazione ripetuta 8 volte

          RET                   ;il controllo ritorna al DOS
PROCEDURA ENDP                  ;fine della procedura

RITARDO   PROC    NEAR
          PUSH    DX            ;salva i valori originali di DX
          PUSH    CX            ;e di CX
          MOV     DX,10         ;10 unità di tempo di attesa in DX
P1:       MOV     CX,0FF00H     ;carica in CX il contatore 0FF00H
P2:       DEC     CX            ;decrementa CX: il tempo passa
          JNZ     P2            ;se diverso da zero, continua
          DEC     DX            ;se CX = 0, decrementa DX
          JNZ     P1            ;se DX è diverso da zero, carica CX di nuovo
          POP     CX            ;se DX = 0, ripristina i valori di CX
          POP     DX            ;e di DX
          RET                   ;definisce il tipo di ritorno
RITARDO   ENDP                  ;fine della procedura NEAR

CODICE    ENDS                  ;fine del segmento di codice
          END                   ;fine del programma
```

Figura 7.5 Programma che illustra l'uso di una procedura NEAR

macro RITARDO (RITARDO 10) è stata sostituita con l'istruzione di chiamata a procedura CALL RITARDO.

```
            MOV    CX,08H      ;ripete il ciclo 8 volte
            MOV    BL,00H      ;colore originale per lo sfondo
ANCORA:     LEA    BP,BACK     ;stringa di caratteri spazio
            MOV    DX,0000     ;cursore in alto a sinistra
            MOV    AH,19       ;attributo di stringa
            MOV    AL,1        ;stampa dei caratteri e aggiornamamento
                               ;cursore
            PUSH   CX          ;salva il contatore
            MOV    CX,07D0H    ;visualizzazione di 2000 caratteri spazio
            INT    10H         ;chiamata di interruzione
            CALL   RITARDO     ;chiamata della procedura NEAR
                               ;RITARDO
            ADD    BL,10H      ;modifica del colore dello sfondo
            POP    CX          ;ripristina il contatore originale
            LOOP   ANCORA      ;operazione ripetuta 8 volte
            RET                ;il controllo ritorna al DOS
PROCEDURA   ENDP               ;fine della procedura
```

La procedura viene invocata con un'istruzione CALL RITARDO, che sostituisce l'istruzione RITARDO 10. Il passaggio dei parametri non avviene come nelle macro: per generare 10 unità di tempo di attesa, il valore 10 viene definito all'interno della procedura RITARDO.
La procedura RITARDO può essere considerata un piccolo programma indipendente:

```
RITARDO     PROC   NEAR
            PUSH   DX          ;salva i valori originali di DX
            PUSH   CX          ;e di CX
            MOV    DX,10       ;10 unità di tempo di attesa in DX
P1:         MOV    CX,0FF00H   ;carica in CX il contatore 00FF00H
P2:         DEC    CX          ;decrementa CX: il tempo passa
            JNZ    P2          ;se diverso da zero, continua
            DEC    DX          ;se CX = 0, decrementa DX
            JNZ    P1          ;se DX è diverso da zero, carica CX di
                               ;nuovo
            POP    CX          ;se DX = 0, ripristina i valori di CX
            POP    DX          ;e di DX
            RET                ;definisce il tipo di ritorno
RITARDO     ENDP               ;fine della procedura NEAR
```

Non sembra esserci grande differenza tra una macro e una procedura di tipo NEAR. Invece, le differenze esistono e sono notevoli. La Figura 7.6 evidenzia il file .LST prodotto dalla traduzione del programma di Figura 7.5. Confrontando questo listato con quello prodotto dalla traduzione del programma di Figura 7.2, si nota subito l'assenza di caratteri +, in quanto ora l'assembler non ha espanso alcuna macro.

```
                        PAGE ,132
                        ;per macchine IBM 8088/80286 con monitor RGB
                        ;il programma illustra l'uso di una procedura NEAR

0000                    STACK     SEGMENT PARA STACK
0000  40 [                        DB      64 DUP ('MYSTACK ')
        4D 59 53 54
        41 43 4B 20
                   ]
0020                    STACK     ENDS

0000                    DATI      SEGMENT PARA 'DATI'
0000 07D0 [             BACK      DB      2000 DUP (' ')
         20
           ]
07D0                    DATI      ENDS


0000                    CODICE    SEGMENT PARA 'CODICE' ;definisce il segmento di codice
0000                    PROCEDURA PROC    FAR         ;inizio della procedura
                                  ASSUME  CS:CODICE,ES:DATI,SS:STACK
0000 1E                           PUSH    DS          ;salva DS sullo stack
0001 2B C0                        SUB     AX,AX       ;azzera AX
0003 50                           PUSH    AX          ;salva 0 sullo stack
0004 B8 ----R                     MOV     AX,DATI     ;indirizzo di DATI in AX
0007 8E C0                        MOV     ES,AX       ;indirizzo di DATI in ES

                       ;questo segmento di programma cancella lo schermo, visualizzando
                       ;80 per 25 caratteri spazio.
                       ;Ripetendo la stessa operazione con un valore differente nel
;registro BL, il colore di tutto lo schermo viene modificato! La
;procedura RITARDO garantisce per un periodo di tempo specificato
;l'immutabilità del colore sullo schermo.
0009 B9 0008                      MOV     CX,08H      ;ripete il ciclo 8 volte
000C B3 00                        MOV     BL,00H      ;colore originale per lo sfondo
000E 8D 2E 0000 R       ANCORA:   LEA     BP,BACK     ;stringa di caratteri spazio
0012 BA 0000                      MOV     DX,0000     ;cursore in alto a sinistra
0015 B4 13                        MOV     AH,19       ;attributo di stringa
0017 B0 01                        MOV     AL,1        ;stampa dei caratteri e aggiornamento cursore
0019 51                           PUSH    CX          ;salva il contatore
001A B9 07D0                      MOV     CX,07D0H    ;visualizzazione di 2000 caratteri spazio
001D CD 10                        INT     10H         ;chiamata di interruzione
001F E8 0029 R                    CALL    RITARDO     ;chiamata della procedura NEAR RITARDO
0022 80 C3 10                     ADD     BL,10H      ;modifica del colore dello sfondo
0025 59                           POP     CX          ;ripristina il contatore originale
0026 E2 D9                        LOOP    ANCORA      ;operazione ripetuta 8 volte

0028 CB                           RET                 ;il controllo ritorna al DOS
0029                    PROCEDURA ENDP                ;fine della procedura

0029                    RITARDO   PROC    NEAR
0029 52                           PUSH    DX          ;salva i valori originali di DX
002A 51                           PUSH    CX          ;e di CX
002B BA 000A                      MOV     DX,TEMPO    ;variabile TEMPO in DX
002E B9 FF00            P1:       MOV     CX,0FF00H   ;carica in CX il contatore 00FF00H
0031 49                 P2:       DEC     CX          ;decrementa CX: il tempo passa
0032 75 FD                        JNZ     P2          ;se diverso da zero, continua
0034 4A                           DEC     DX          ;se CX = 0, decrementa DX
0035 75 F7                        JNZ     P1          ;se DX è diverso da zero, carica CX di nuovo
0037 59                           POP     CX          ;se DX = 0, ripristina i valori di CX
```

```
0038 5A                     POP     DX          ;e di DX
0039 C3                     RET                 ;definisce il tipo di ritorno
003A            RITARDO     ENDP                ;fine della procedura NEAR

003A            CODICE      ENDS                ;fine del segmento di codice

                            END                 ;fine del programma

Segments and Group:

            Name            Size  Align  Combine   Class

CODICE..................    0036  PARA   NONE      'CODICE'
DATI....................    07D0  PARA   NONE      'DATI'
STACK...................    0200  PARA   STACK

Symbols:

            Name            Type   Value  Attr

ANCORA..................    L NEAR 000E   CODICE
BACK....................    L BYTE 0000   DATI    Length =07D0
RITARDO.................    N PROC 0029   CODICE  Length =0011
PROCEDURA...............    F PROC 0000   CODICE  Length =0029
P1......................    L NEAR 002E   CODICE
P2......................    L NEAR 0031   CODICE

50092 Bytes free

Warning Severe
Errors  Errors
0       0
```

Figura 7.6 File .LST prodotto dalla traduzione del programma di Figura 7.5

Quando l'assembler incontra la procedura RITARDO, la traduce come se si trattasse di un segmento di codice qualsiasi, mentre la presenza di una macro obbliga l'assembler a duplicarne il codice ogni volta che viene invocata nel programma. Una procedura può essere chiamata un numero qualunque di volte tramite l'istruzione CALL.
Il codice della procedura RITARDO appare una sola volta nel programma, indipendentemente dal numero di volte che viene chiamata. Si può notare che il codice macchina per l'istruzione RET di tipo FAR è 0CBH, mentre il codice macchina per l'istruzione RET di tipo NEAR è 0C3H.
Poiché le procedure non vengono duplicate ogni volta che sono invocate, i programmi che utilizzano le procedure risultano quindi più compatti. L'esecuzione però non procede in sequenza, per cui si verifica una perdita di tempo dovuta al passaggio del controllo dal programma principale alla procedura e, se l'istruzione di chiamata è interna ad un ciclo, l'incremento del tempo di esecuzione diventa rilevante.

LIBRERIA DI PROCEDURE

È possibile raggruppare un insieme di procedure in una libreria e poi invocarle da programma. Queste procedure sono di tipo FAR in quanto il segmento di codice da cui vengono chiamate non coincide con il segmento di codice in cui sono definite. La Figura 7.7 mostra un programma, molto simi-

```
;per macchine IBM 8088/80286
;programma che illustra la chiamata di una procedura FAR

          EXTRN   PUSHA:FAR,POPA:FAR,RITARDO:FAR,CANCELLA:FAR
          EXTRN   STAMPANUM:FAR,CURSORE:FAR,STAMPACHAR:FAR

STACK     SEGMENT PARA STACK
          DB      48 DUP ('STACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
MESSAGGIO DB      'Semplice programma di conteggio'
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE' ;definisce il segmento di codice
PROCEDURA PROC    FAR           ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS            ;salva DS sullo stack
          SUB     AX,AX         ;azzera AX
          PUSH    AX            ;salva 0 sullo stack
          MOV     AX,DATI       ;indirizzo di DATI in AX
          MOV     DS,AX         ;indirizzo di DATI in DS

          CALL    CANCELLA      ;chiamata della procedura CANCELLA
          MOV     DX,19H        ;posizione del cursore
          CALL    CURSORE       ;messaggio al centro dello schermo
          LEA     DX,MESSAGGIO  ;testo da visualizzare
          CALL    STAMPACHAR    ;visualizzazione del messaggio

          MOV     AX,00         ;contatore inizializzato a zero
ANCORA:   MOV     DX,0C28H      ;centro dello schermo
          CALL    CURSORE       ;posizionamento del cursore
          CALL    STAMPANUM     ;visualizza il contenuto di AL sullo schermo
          MOV     DX,10H        ;attesa di 10 istanti di tempo
          CALL    RITARDO       ;attesa
          ADD     AL,01         ;somma di una unità al registro AL
          DAA                   ;sistemazione decimale del risultato
          CMP     AL,50H        ;dopo cinque cicli, fine
          JE      FINE
          JMP     ANCORA        ;ciclo successivo
FINE:     CALL    CANCELLA      ;ulteriore cancellazione dello schermo

          RET                   ;il controllo ritorna al DOS
PROCEDURA ENDP                  ;fine della procedura
CODICE    ENDS                  ;fine del segmento di codice

          END                   ;fine del programma
```

Figura 7.7 Programma che chiama alcune procedure di tipo FAR

le a quello presentato in Figura 7.4, che utilizza la direttiva EXTRN per referenziare correttamente le procedure esterne:

```
EXTRN    PUSHA:FAR,POPA:FAR,RITARDO:FAR,CANCELLA:FAR
EXTRN    STAMPANUM:FAR,CURSORE:FAR,STAMPACHAR:FAR
```

Il nome del file, in cui queste procedure sono memorizzate e che viene collegato al programma principale dal linker, non pone alcun requisito particolare.
Riportiamo qui di seguito una parte del programma di Figura 7.7:

```
CALL     CANCELLA          ;chiamata della procedura CANCELLA
MOV      DX,19H            ;posizione del cursore
CALL     CURSORE           ;messaggio al centro dello schermo
LEA      DX,MESSAGGIO      ;testo da visualizzare
CALL     STAMPACHAR        ;visualizzazione del messaggio
```

Confrontate queste istruzioni di codice con le analoghe istruzioni del programma di Figura 7.4. Si noti come il registro DX sia inizializzato con la posizione del cursore, prima che venga invocata la procedura CURSORE. Allo stesso modo, prima della chiamata della procedura STAMPACHAR, viene memorizzato in DX l'indirizzo di MESSAGGIO. Questa tecnica risolve il problema del passaggio dei parametri: i parametri vengono memorizzati in alcuni registri, che possiedono una visibilità globale.
La Figura 7.8 mostra il contenuto di un file di codice sorgente in cui sono memorizzate alcune procedure che vengono chiamate da altri moduli di programma. Confrontate questa figura con la Figura 7.3.
Il numero di direttive che occorrono per dichiarare le procedure esterne è minimo:

```
CODICE   SEGMENT
         PUBLIC                PUSHA,POPA,RITARDO,CANCELLA
         STAMPANUM,CURSORE,STAMPACHAR
         ASSUME                CS:CODICE
```

(Qui si seguito si inserisce il codice delle procedure)

```
CODICE   ENDS
         END
```

In questo caso, ogni procedura viene dichiarata PUBLIC in quanto viene referenziata da altri moduli di programma. Questa direttiva permette dunque uno scambio corretto di informazioni tra segmenti di codice distinti.
La procedura CANCELLA, che riportiamo qui di seguito, può essere presa come esempio per indicare la struttura di una generica procedura:

```
CANCELLA    PROC    FAR          ;cancella lo schermo a colori
            PUSH    AX           ;salva i registri
```

```
CODICE   SEGMENT

         PUBLIC PUSHA,POPA,RITARDO,CANCELLA,STAMPANUM,CURSORE,STAMPACHAR

         ASSUME CS:CODICE

PUSHA    PROC   FAR          ;salva i registri macchina
         PUSH   AX           ;non è necessario su 80286/80386
         PUSH   CX
         PUSH   DX
         PUSH   BX
         PUSH   SP
         PUSH   BP
         PUSH   SI
         PUSH   DI
         RET
PUSHA    ENDP

POPA     PROC   FAR
         POP    DI           ;ripristina i registri macchina
         POP    SI           ;non è necessario su 80286/80386
         POP    BP
         POP    SP
         POP    BX
         POP    DX
         POP    CX
         POP    AX
         RET
POPA     ENDP

RITARDO  PROC   FAR          ;attesa a tempo controllata software
         PUSH   CX           ;salva i registri
         PUSH   DX
P1:      MOV    CX,0FF00H    ;tempo di ogni attesa in CX
P2:      DEC    CX           ;decrementa CX
         JNZ    P2           ;se non è zero, P2
         DEC    DX           ;se è zero, decrementa DX
         JNZ    P1           ;se DX non è zero, P1
         POP    DX           ;ripristino dei registri
         POP    CX
         RET
RITARDO  ENDP

CANCELLA PROC   FAR          ;cancella lo schermo a colori
         PUSH   AX           ;salva i registri
         PUSH   BX
         PUSH   CX
         PUSH   DX
         MOV    CX,0         ;angolo superiore della finestra
         MOV    DX,2476H     ;angolo inferiore della finestra
         MOV    BH,7         ;attributo di schermo (normale)
         MOV    AX,0600H     ;parametri di interruzione
         INT    10H          ;chiamata di interruzione
         POP    DX           ;ripristino dei registri
         POP    CX
         POP    BX
         POP    AX
         RET
CANCELLA ENDP
```

```
STAMPANUM  PROC   FAR       ;visualizza un numero ASCII sullo schermo
           PUSH   AX        ;salva i registri
           PUSH   CX
           PUSH   AX        ;salva AX perchè sarà modificato
           MOV    AH,15     ;stato di visualizzazione
           INT    10H       ;chiamata di interruzione, uscita in BX
           POP    AX        ;ripristina il valore di AX
           AND    AL,0FH    ;azzera i quattro bit più significativi
           OR     AL,30H    ;conversione ASCII
           MOV    AH,10     ;parametro di interruzione
           MOV    CX,1      ;numero di caratteri da visualizzare
           INT    10H       ;chiamata di interruzione
           POP    CX        ;ripristino dei registri
           POP    AX
STAMPANUM  ENDP

CURSORE    PROC   FAR       ;posiziona il cursore in LOCAZIONE
           PUSH   AX        ;salva i registri
           MOV    AH,15     ;stato di visualizzazione
           INT    10H       ;chiamata di interruzione, uscita in BX
           MOV    AH,2      ;parametro relativo al cursore
           INT    10H       ;chiamata di interruzione
           POP    AX        ;ripristino dei registri
           RET
CURSORE    ENDP

STAMPACHAR PROC   FAR       ;visualizza una stringa di caratteri
           PUSH   AX        ;La stringa viene passata alla macro
           MOV    AH,9      ;La visualizzazione inizia dalla posizione
           INT    21H       ;corrente del cursore e procede fino a
           POP    AX        ;quando si incontra il tappo '$'.
STAMPACHAR ENDP
```

Figura 7.8 Insieme di procedure che possono essere chiamate da un programma utente

```
           PUSH   BX
           PUSH   CX
           PUSH   DX
           MOV    CX,0        ;angolo superiore della finestra
           MOV    DX,2476H    ;angolo inferiore della finestra
           MOV    BH,7        ;attributo di schermo (normale)
           MOV    AX,0600H    ;parametri di interruzione
           INT    10H         ;chiamata di interruzione
           POP    DX          ;ripristino dei registri
           POP    CX
           POP    BX
           POP    AX
           RET
CANCELLA   ENDP
```

La dichiarazione delle procedure di Figura 7.8 obbliga a specificare il nome, la direttiva PROC e l'attributo FAR. Ogni procedura termina con l'istruzione RET, seguita dal nome della procedura e dalla direttiva ENDP. Il corpo

di ogni procedura è stato modificato lievemente rispetto al codice che abbiamo presentato in Figura 7.3.
In conclusione, disponiamo di due programmi distinti: il primo è il programma principale, mentre il secondo è una libreria di procedure. Ogni programma viene tradotto separatamente dall'assembler e le versioni in codice oggetto così generate (file .OBJ) vengono collegate insieme da un linker. I manuali DOS Microsoft e IBM forniscono una dettagliata descrizione di come realizzare queste operazioni utilizzando diverse opzioni di traduzione e di collegamento. Ad esempio, se il linker risiede nel drive A e i file da collegare si trovano nel drive B, scriviamo:

```
B>A:LINK MAINFILE.OBJ + PROCFILE.OBJ
```

Si ricordi che una libreria di macro è una collezione di routine, non tradotte in codice oggetto, che risiedono in un file sorgente, mentre una libreria di procedure è una collezione di routine, tradotte in codice oggetto, che risiedono in un file oggetto. Le macro vengono incluse nel programma principale quando questo viene tradotto in codice oggetto (comando MASM), mentre le procedure vengono incluse nel programma principale in fase di collegamento (comando LINK).

7.3 Librerie di collegamento

Una libreria di collegamento che viene referenziata dal linker differisce poco da una libreria di procedure, in quanto ogni libreria di collegamento fornisce codice già corretto, che viene utilizzato da altri programmi.
Quando il linker si appresta a collegare i diversi moduli di cui si compone un programma, appare sullo schermo il seguente menu:

```
IBM Personal Computer Linker
Version 2.30 (C) Copyright IBM Corp. 1981, 1985

Object Modules [.OBJ]: B:PROG
Run File [PROG.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
```

L'opzione Libraries del Linker DOS permette all'utente di specificare il nome di una libreria di collegamento. Queste librerie, che vengono appunto referenziate in fase di collegamento, contengono un insieme di procedure in codice oggetto. Utilizzando la versione 2.0 dell'Assemblatore IBM o la versione 3.0 dell'Assemblatore Microsoft, è possibile disporre di un gestore di libreria (Library Manager) che permette all'utente di potenziare una libreria con l'aggiunta di nuove routine. Quando il programma ha bisogno di una

libreria, il nome di quest'ultima deve essere specificato in fase di collegamento in corrispondenza dell'opzione di libreria. L'uso di queste librerie riduce così sensibilmente il tempo che occorre per sviluppare un programma, in quanto le procedure che vengono richieste sono già state scritte e corrette.
Il Library Manager, a tutt'oggi, supporta le seguenti operazioni: aggiunge altri file oggetto oltre a quelli disponibili, cancella file oggetto, estrae un modulo oggetto, rimpiazza un modulo oggetto, crea nuove librerie, stampa il contenuto di una libreria e modifica la dimensione della pagina utilizzata per memorizzare il contenuto di una libreria.
Per dimostrare come creare una libreria, utilizziamo gli esempi già presentati nel precedente paragrafo. Le Figure 7.7 e 7.8 rimangono identiche: la Figura 7.7 mostra il programma principale, mentre la Figura 7.8 contiene la libreria di procedure. Per convertire questa libreria in una libreria di collegamento, possiamo utilizzare il Library Manager. Nel nostro caso, la libreria che andiamo a definire prende il nome di NEWLIB:

```
IBM Personal Computer Library Manager
Version 1.00
(C) Copyright IBM Corp 1984
(C) Copyright Microsoft Corp 1984

Library Name: B:NEWLIB
Library does not exist. Create? Y
Operations: +FIG7-8
List file: NEWLIB.LSF
```

Una volta che la libreria è stata creata, è possibile ampliarla con altre procedure e routine utilizzando il programma Library Manager. In fase di collegamento, indicate semplicemente NEWLIB quando vi viene chiesto il nome della libreria.
Il file NEWLIB.LSF (library LiSt File) contiene le seguenti informazioni:

```
CANCELLA...............FIG7-8 CURSORE............FIG7-8
RITARDO................FIG7-8 POPA...............FIG7-8
STAMPACHAR.............FIG7-8 STAMPANUM..........FIG7-8
PUSHA..................FIG7-8
FIG7-8 Offset: 200H Code and data size: 5E
CANCELLA CURSORE RITARDO POPA
STAMPACHAR STAMPANUM PUSHA
```

Si può immediatamente notare che la libreria di collegamento non differisce molto da una libreria di procedure, anche se esistono alcune differenze ben precise. In primo luogo, si ricordi che una libreria di procedure viene unita agli altri moduli oggetto in fase di collegamento. In secondo luogo, utilizzando il programma Library Manager, è possibile aggiungere nuove procedure a quelle esistenti o eliminarne delle vecchie dalla libreria in qualunque momento. Questa possibilità rende la libreria di collegamento più flessibile rispetto a quelle che abbiamo discusso in precedenza.

7.4 Vantaggi e svantaggi

Le macro, le procedure e le librerie hanno alcuni aspetti in comune: contengono ad esempio parti di codice già corrette, che possono essere successivamente invocate da qualsiasi programma. Questo rende la programmazione più modulare e più efficiente.
Ognuna delle soluzioni che abbiamo discusso nei paragrafi precedenti possiede comunque vantaggi e svantaggi.

Vantaggi delle macro
1. Le macro garantiscono una rapida esecuzione del codice, in quanto vengono eseguite in sequenza.
2. Gli argomenti di una macro influiscono direttamente sul suo operato.
3. Le macro possono essere memorizzate facilmente in una libreria di codice sorgente.
4. Le istruzioni che definiscono l'ambiente di lavoro a macro sono poche e semplici; per una libreria di macro è sufficiente utilizzare le direttive IF1....ENDIF.

Svantaggi delle macro
1. Le macro incrementano il codice sorgente, in quanto vengono espanse ogni volta che il programma le invoca.

Vantaggi delle librerie di procedure
1. Le procedure ottimizzano la lunghezza di codice sorgente, in quanto non vengono espanse quando sono invocate.

Svantaggi delle librerie di procedure
1. Le procedure aumentano il tempo di esecuzione dei programmi, in quanto ad ogni istruzione di chiamata (CALL), il microprocessore deve abbandonare il codice principale e saltare in un'altra parte di codice.
2. Le istruzioni che definiscono l'ambiente di lavoro a procedure sono meno semplici di quelle che occorrono per definire l'ambiente di lavoro a macro. Ogni procedura deve essere dichiarata NEAR oppure FAR e ogni riferimento a procedure di libreria deve essere dichiarato di tipo EXTRN.
3. I parametri passati ad una procedura non possono alterare il suo funzionamento.

Vantaggi delle librerie di collegamento
1. Stessi vantaggi riscontrati per una libreria di procedure.
2. Utilizzando il programma Library Manager, è possibile aggiungere o eliminare routine dalla libreria.

Svantaggi delle librerie di collegamento
1. Stessi svantaggi riscontrati per una libreria di procedure.

In conclusione
Per valutare quindi quale tra le soluzioni che abbiamo presentato sia la migliore, tenete presente le seguenti considerazioni:
1. Nel caso di routine brevi, utilizzare la soluzione a macro (operazioni più rapide, con piccolo incremento di codice).
2. Nel caso di routine che non vengono invocate frequentemente da un programma, utilizzare la soluzione a macro (se la routine non viene chiamata frequentemente, l'espansione di codice non compromette negativamente l'esecuzione del programma).
3. Se state scrivendo o sviluppando un programma, utilizzate la soluzione a macro (le macro sono più semplici da scrivere, memorizzare e chiamare).
4. Nel caso di routine di grandi dimensioni, utilizzate la soluzione a procedure (ottimizzazione della lunghezza del codice sorgente).
5. Nel caso di routine che vengono invocate frequentemente da un programma, utilizzate la soluzione a procedure (le procedure non vengono espanse all'interno del programma).
6. Se le stesse routine vengono utilizzate frequentemente nei vostri programmi, memorizzatele in una libreria di collegamento così da ottimizzare l'accesso.

Alcuni programmatori in linguaggio assemblatore non utilizzano mai le macro, mentre altri cercano di evitare di scrivere procedure. La scelta di quale tecnica programmativa utilizzare deve pertanto basarsi sulle esigenze che si presentano di volta in volta.

8

Tecniche avanzate di programmazione

Gli esempi discussi nei precedenti capitoli, anche se non presentano eccessive difficoltà programmative, hanno permesso di acquisire una certa familiarità con le istruzioni del linguaggio assemblatore.
I programmi che presentiamo in questo capitolo, invece, risultano sicuramente più complessi e inizialmente meno interessanti dei precedenti per un principiante. Esempi tipici di applicazioni che vengono qui trattate riguardano la realizzazione di menu, che permettono all'utente di dialogare da tastiera con il calcolatore, la lettura e la scrittura di dati su dischetto, ecc.
Lo scopo principale di questo capitolo è quello di introdurre gradualmente nuovi concetti della programmazione in linguaggio assemblatore e di indicare le tecniche più adatte per risolvere i problemi che si presentano. Ad esempio, è possibile, con lievi modifiche, fare in modo che un programma che realizza una rubrica telefonica su file sia in grado di memorizzare i dati esaminati da un lettore digitale-analogico (D/A) oppure calcolati da una equazione.
Ognuno dei primi cinque programmi di questo capitolo affronta un differente problema di programmazione e lo risolve adottando diverse soluzioni, allo scopo di insegnare le tecniche di programmazione più adatte ed efficienti.

8.1 Visualizzazione di un grafo sullo schermo a colori

Problema 1: Visualizzare un'onda sinusoidale su uno schermo ad alta risoluzione grafica.

Nel campo scientifico, una delle più funzioni più utili che un calcolatore può

eseguire è quella di visualizzare graficamente sullo schermo i risultati di operazioni o di elaborazioni numeriche. Come anticipazione ai programmi più complessi che verranno discussi nel Capitolo 9 e che utilizzeranno le funzioni supportate dai coprocessori matematici 80287/80387, l'esempio che presentiamo ora esegue il tracciato di un'onda sinusoidale su uno schermo grafico a colori ad alta risoluzione 640 × 200, servendosi di una tabella di lookup. Una delle maggiori limitazioni della programmazione in linguaggio assemblatore consiste nella mancanza di comandi matematici, per cui il programmatore è costretto a codificare esplicitamente le operazioni di estrazione di radice quadrata e cubica, di integrazione, di derivazione, di calcolo trigonometrico e di analisi statistica. Spesso, però, gli algoritmi che realizzano queste operazioni sono difficili da scrivere e soprattutto da implementare. Nel nostro caso, la funzione seno (come altre funzioni trigonometriche) non è supportata dai microprocessori 80286/80386, ma il programmatore può approssimarla con uno sviluppo in serie, che però, essendo costituito da potenze e fattoriali, risulta non facile da implementare in linguaggio assemblatore.
Un'onda sinusoidale viene visualizzata sullo schermo con una successione di punti luminosi, le cui posizioni sono definite dal valore delle coordinate orizzontali e verticali (X,Y). L'insieme di questi valori può essere calcolato prima di scrivere il programma e memorizzato in una tabella. Il programma, in questo modo, deve solo accedere alla tabella per referenziare i singoli punti di cui si compone l'onda sinusoidale, evitando così calcoli matematici piuttosto complessi.
Riportiamo qui di seguito parte del segmento di dati che è stato definito nel programma:

```
SENO    DB      00,02,04,05,07,09,11,12,14,16,17,19,21,23,24,26
        DB      28,29,31,33,34,36,38,39,41,42,44,45,47,49,50
        DB      52,53,55,56,57,59,60,62,63,64,66,67,68,70,71
        DB      72,73,74,76,77,78,79,80,81,82,83,84,85,86,87
        DB      88,88,89,90,91,91,92,93,93,94,95,95,96,96,97
        DB      97,97,98,98,99,99,99,99,100,100,100,100,100,100,100
```

SENO è il nome della tabella di lookup e i numeri in essa contenuti forniscono i valori (moltiplicati per 100 e arrotondati alla seconda o terza cifra, per essere adattati alla risoluzione dello schermo) della funzione seno per angoli compresi tra 0 e 90 gradi. In entrambe le modalità di risoluzione (media e alta), la risoluzione verticale del monitor a colori IBM è di 200 pixel. Se l'onda sinusoidale deve avere una ampiezza di +100, è indispensabile rapportare il seno di 90 gradi (che vale 1) al valore 100. Poiché il grafico copre un'angolazione di 360 gradi ed è simmetrico, la tabella SENO sottintende anche le informazioni relative al secondo, terzo e quarto quadrante.
Il programma, in definitiva, effettua le seguenti operazioni:

1. cerca un valore nella tabella SENO a cui associa la coordinata Y sullo schermo;

2. il grafico appare sullo schermo da sinistra a destra, per cui viene incrementata di una unità (pixel) la posizione orizzontale per ogni punto verticale visualizzato. Ci sono 360 punti da visualizzare (360 gradi) e 640 pixel orizzontali (cioè posizioni sullo schermo ad alta risoluzione).

L'informazione contenuta nella tabella SENO è sufficiente per eseguire il grafico sui quattro quadranti. La Figura 8.1 mostra l'intero programma, che contiene la definizione di due macro (PREPARA e VISUALIZZA).
Riportiamo qui di seguito la macro PREPARA, che inizializza lo schermo ad una grafica ad alta risoluzione:

```
PREPARA   MACRO              ;;schermo ad alta risoluzione
          MOV  AH,00         ;;200 per 640 pixel B/N
          MOV  AL,06
          INT  10H
          ENDM
```

Le opzioni che sono disponibili per l'interruzione INT 10H sono state discusse nel Capitolo 5. Esaminiamo più dettagliatamente la macro VISUALIZZA:

```
VISUALIZZA MACRO             ;;visualizza punti sullo schermo
          MOV  AH,12
          MOV  AL,01
          MOV  CX,ANGOLO
          ADD  CX,140        ;;immagine al centro
          MOV  DH,00
          MOV  DL,TEMP
          INT  10H
          ENDM
```

In questa macro viene invocata l'interruzione 10H. Il registro CX memorizza la posizione orizzontale, mentre il registro DX memorizza la posizione verticale in cui deve essere visualizzato il punto corrente del grafico. Poiché sono definite complessivamente 640 posizioni orizzontali distinte e gli angoli non superano i 360 gradi, per visualizzare al centro dello schermo l'onda sinusoidale, utilizziamo un offset di 140 riferito al margine sinistro dello schermo ($640-360=280$, e $280/2=140$). La variabile TEMP contiene il valore della posizione verticale di visualizzazione (compresa tra 0 e 100) e, quando viene invocata la macro VISUALIZZA, le variabili ANGOLO e TEMP stabiliscono dove deve essere visualizzato il punto luminoso.
Riportiamo qui di seguito la parte principale del codice che visualizza i punti sullo schermo:

```
          PREPARA            ;schermo grafico 200 per 640
ANCORA:   LEA  BX,SENO       ;inizio della tabella
          MOV  AX,ANGOLO     ;valore dell'angolo in AX
```

```
;per macchine 8088/80386
;programma che utilizza una tabella di lookup, contenente i
;valori della funzione seno, per tracciare il grafico di un'onda
;sinusoidale, servendosi di interruzione BIOS.

PREPARA  MACRO                      ;;schermo ad alta risoluzione
         MOV      AH,00             ;;200 per 640 pixel B/N
         MOV      AL,06
         INT      10H
         ENDM

VISUALIZZA  MACRO                   ;;visualizza punti sullo schermo
         MOV      AH,12
         MOV      AL,01
         MOV      CX,ANGOLO
         ADD      CX,140            ;;immagine al centro
         MOV      DH,00
         MOV      DL,TEMP
         INT      10H
         ENDM

STACK    SEGMENT PARA STACK
         DB       48 DUP ('STACK ')
STACK    ENDS

DATI     SEGMENT PARA 'DATI'
SENO     DB       00,02,04,05,07,09,11,12,14,16,17,19,21,23,24,26
         DB       28,29,31,33,34,36,38,39,41,42,44,45,47,49,50
         DB       52,53,55,56,57,59,60,62,63,64,66,67,68,70,71
         DB       72,73,74,76,77,78,79,80,81,82,83,84,85,86,87
         DB       88,88,89,90,91,91,92,93,93,94,95,95,96,96,97
         DB       97,97,98,98,99,99,99,99,100,100,100,100,100,100,100
ANGOLO   DW       0
TEMP     DB       0
DATI     ENDS

CODICE   SEGMENT PARA 'CODICE' ;definisce il segmento di codice
PROCEDURA PROC    FAR          ;inizio della procedura
         ASSUME   CS:CODICE,DS:DATI,SS:STACK
         PUSH     DS           ;salva DS sullo stack
         SUB      AX,AX        ;azzera AX
         PUSH     AX           ;salva 0 sullo stack
         MOV      AX,DATI      ;indirizzo di DATI in AX
         MOV      DS,AX        ;indirizzo di DATI in DS

;esempio di utilizzo di una tabella di lookup per determinare il
;tracciato dell'onda sinusoidale. I valori del seno sono stati
;moltiplicati per 100 e arrotondati ad un numero intero
         PREPARA               ;schermo grafico 200 per 640
ANCORA:  LEA      BX,SENO      ;inizio della tabella
         MOV      AX,ANGOLO    ;valore dell'angolo in AX
         CMP      AX,180       ;è maggiore di 180 gradi?
         JLE      NEWQUAD      ;se è minore, siano nel 1° o nel 2° quadrante
         SUB      AX,180       ;altrimenti, correzione dell'angolo
NEWQUAD: CMP      AX,90        ;è maggiore di 90 gradi?
         JLE      SECQUAD      ;se è maggiore, siamo nel 2° quadrante
         NEG      AX           ;angolo di valore negativo
         ADD      AX,180       ;valore corretto, se l'angolo non è minore di 90 gradi
SECQUAD: ADD      BX,AX        ;somma di offset in BX
         MOV      AL,SENO[BX]  ;il valore del seno è in AL
```

```
        CMP     ANGOLO,180      ;se l'angolo non è minore di 180, aggiorna la posizione sullo schermo
        JGE     SPIAZZ
        NEG     AL              ;altrimenti, angolo di valore negativo
        ADD     AL,100          ;somma di 100 al valore, per posizionarsi
        JMP     PRONTO          ;correttamente sullo schermo
SPIAZZ: ADD     AL,99
PRONTO: MOV     TEMP,AL         ;valore corretto del seno in TEMP
        VISUALIZZA              ;chiamata della macro che visualizza il punto
        ADD     ANGOLO,1        ;prossimo angolo
        CMP     ANGOLO,360      ;siamo arrivati a 360 gradi?
        JLE     ANCORA          ;se no, ripetere l'operazione
;fine dell'esempio di utilizzo di una tabella di lookup per
;tracciare un'onda sinusoidale

;attesa di un comando da tastiera prima di riattivare la modalità
;di visualizzazione di testi sullo schermo
        MOV     AH,07           ;parametro per tastiera
        INT     21H             ;lettura tastiera
        MOV     AH,00           ;parametro per schermo
        MOV     AL,03           ;modalità colore 25 per 80
        INT     10H             ;aggiornamento visualizzazione di schermo

        RET                     ;il controllo ritorna al DOS
PROCEDURA ENDP                  ;fine della procedura
CODICE  ENDS                    ;fine del segmento di codice

        END                     ;fine del programma
```

Figura 8.1 Programma che utilizza una tabella di lookup per tracciare il grafico di un'onda sinusoidale

```
          CMP  AX,180     ;è maggiore di 180 gradi?
          JLE  NEWQUAD    ;se è minore, siano nel 1° o nel 2° quadrante
          SUB  AX,180     ;altrimenti, correzione dell'angolo
NEWQUAD:  CMP  AX,90      ;è maggiore di 90 gradi?
          JLE  SECQUAD    ;se è maggiore, siamo nel 2° quadrante
          NEG  AX         ;angolo di valore negativo
          ADD  AX,180     ;valore corretto, se l'angolo non è minore di
                          ;90 gradi
```

È indispensabile determinare in quale quadrante si trovi l'angolo, per poter ricavare dalla tabella di lookup il corretto valore del seno. Innanzitutto, il valore dell'angolo, che è contenuto in AX, viene confrontato con il valore 180 (gradi): se l'angolo non è maggiore di questo valore, si tratta di un angolo positivo che appartiene al primo o al secondo quadrante. Se invece l'angolo è maggiore di 180, cioè appartiene al terzo o al quarto quadrante, il valore della funzione seno è negativo.
Inizialmente, all'angolo viene sottratto 180 (gradi). Poi, il programma verifica se il valore dell'angolo così ottenuto è maggiore di 90 (gradi). Se lo è, l'angolo viene sottratto al valore 180. Questa operazione non è immediata, in quanto non è lecito utilizzare l'istruzione SUB 180,AX (questa istruzione, in-

fatti, non specifica dove memorizzare il risultato della sottrazione), per cui il contenuto di AX viene complementato e poi viene sommato a 180. Il codice che viene referenziato dall'etichetta SECQUAD determina, sommando il contenuto dei registri AX e BX, il valore dello spiazzamento che permette di accedere correttamente agli elementi della tabella SENO, come viene indicato qui di seguito:

```
SECQUAD: ADD   BX,AX          ;somma di offset in BX
         MOV   AL,SENO[BX]    ;il valore del seno è in AL
         CMP   ANGOLO,180     ;se l'angolo non è minore di 180,
                              ;aggiorna la posizione sullo schermo
         JGE   SPIAZZ
         NEG   AL             ;altrimenti, angolo di valore negativo
         ADD   AL,100         ;somma di 100 al valore, per posizionarsi
         JMP   PRONTO         ;correttamente sullo schermo
SPIAZZ:  ADD   AL,99
PRONTO:  MOV   TEMP,AL        ;valore corretto del seno in TEMP
         VISUALIZZA           ;chiamata della macro che visualizza
                              ;il punto
         ADD   ANGOLO,1       ;prossimo angolo
         CMP   ANGOLO,360     ;siamo arrivati a 360 gradi?
         JLE   ANCORA         ;se no, ripetere l'operazione
```

Le seguenti cinque linee di codice indicano come devono essere visualizzati i punti sullo schermo. La coordinata verticale di inzio schermo è 0, mentre la coordinata verticale di fine schermo è 199 (l'ampiezza verticale picco picco dell'onda vale quindi a 200). Un'ampiezza 0 per l'onda sinusoidale corrisponde ad una posizione verticale sullo schermo pari a 100. Quando la macro VISUALIZZA è stata invocata, il valore contenuto in ANGOLO viene incrementato e poi controllato per verificare se tutti i punti sono stati visualizzati. Se la verifica ha esito negativo, il processo di visualizzazione continua, altrimenti il programma passa ad eseguire la successiva sezione di codice.

```
MOV   AH,07    ;parametro per tastiera
INT   21H      ;lettura tastiera
```

Queste due istruzioni codificano la chiamata di una interruzione DOS (si consulti il Capitolo 5, per avere maggiori dettagli a riguardo), attraverso cui il programma attende un comando da tastiera. In questo modo, il programmatore può osservare sullo schermo il grafico visualizzato, prima che l'immagine venga cancellata quando lo schermo ritorna in modalità normale.

```
MOV   AH,00    ;parametro per schermo
MOV   AL,03    ;modalità colore 25 per 80
INT   10H      ;aggiornamento visualizzazione di schermo
```

Da ultimo, quando viene premuto un tasto, l'interruzione BIOS restituisce all'utente uno schermo 25 × 80 in modalità normale.

L'uso delle tabelle di lookup è particolarmente adatto per visualizzare sullo schermo una grande quantità di funzioni matematiche, adattando i valori delle coordinate con quelle dello schermo. Una volta che è stata eseguita questa operazione, i valori risultanti possono essere memorizzati in una tabella di consultazione e richiamati in ogni momento dal programma. (Poiché la tabella di lookup contiene valori predefiniti, la forma d'onda che è possibile visualizzare è sempre la stessa – a meno di modifiche sui rapporti di scala – per cui il programma risulta poco flessibile). Questa tecnica garantisce di solito una maggiore velocità rispetto alla soluzione che calcola e visualizza sullo schermo i punti del grafico durante l'esecuzione del programma (se il programmatore è interessato a questa soluzione, consulti il Capitolo 9 in cui vengono utilizzate le funzioni matematiche che sono disponibili sul coprocessore 80287/80387).

8.2 Programma che esegue un conteggio in secondi

Problema 2: Scrivere un programma che visualizza il tempo trascorso. Il conteggio deve iniziare quando il programma parte e deve essere a sei cifre. Il passo di avanzamento del conteggio è di un secondo.

Questo problema presenta alcune difficoltà: come visualizzare il conteggio (in secondi); dove visualizzare il conteggio sullo schermo; come organizzare il processo di conteggio (nel formato ASCII, BCD o binario); come mantenere l'accurattezza del conteggio.
Facciamo le seguenti ipotesi: il programma visualizza al centro dello schermo un conteggio decimale a sei cifre, la cui accuratezza viene mantenuta attraverso l'hardware del calcolatore; si utilizza la somma decimale di numeri compattati (DAA) piuttosto che la somma ASCII di numeri non compattati (AAA); le sei cifre decimali del conteggio vengono memorizzate in tre variabili: SEC12, SEC34 e SEC56. (In alternativa, si può utilizzare una variabile di tipo DD, cioè doubleword, accessibile tramite l'operatore BYTE PTR, ma questa scelta complica il programma).
La Figura 8.2 mostra l'intero programma, in cui viene inclusa la libreria di macro MACLIB.MAC, che abbiamo definito nel precedente capitolo e che deve risiedere nel drive C. Inoltre, vengono utilizzate due ulteriori macro: RT, per avere un accurato conteggio in secondi, e SCHERMO, per visualizzare sullo schermo il conteggio decimale.
Riportiamo qui di seguito la macro RT:

```
RT          MACRO   ATTESA      ;;tempo di attesa (1 – 60 sec) che viene
            LOCAL   ANCORA      ;;passato alla macro tramite ATTESA
            PUSH    AX          ;;salva i registri
```

```
;programma per macchine 8088/80386
;programma che esegue il conteggio in secondi da 0 a 999999 e lo
;riazzera. Viene inclusa la libreria MACLIB.MAC in fase di
;traduzione in codice oggetto
;
PAGE ,132                           ;dimensione pagina 66 per 132

IF1                                 ;include una libreria di macro
         INCLUDE C:MACLIB.MAC
ENDIF

RT       MACRO   ATTESA             ;;tempo di attesa (1-60 sec) che viene
         LOCAL   ANCORA             ;;passato alla macro tramite ATTESA
         PUSH    AX                 ;;salva i registri
         PUSH    BX
         PUSH    CX
         PUSH    DX
         MOV     AH,2CH             ;;lettura ora corrente
         INT     21H
         MOV     BH,DH              ;;somma del ritardo ai secondi correnti
         ADD     BH,ATTESA
         CMP     BH,60              ;;aggiornamento non oltre i 60 secondi
         JL      ANCORA
         SUB     BH,60
ANCORA:  MOV     AH,2CH             ;;rilettura dell'ora
         INT     21H
         CMP     BH,DH              ;;è finito il ritardo?
         JNE     ANCORA
         POP     DX                 ;;ripristina i registri
         POP     CX
         POP     BX
         POP     AX
         ENDM

SCHERMO  MACRO   TEMPO,POS          ;;visualizza le cifre sullo schermo
         PUSH    AX
         PUSH    CX
         MOV     AL,TEMPO           ;;cifra da visualizzare
         AND     AL,0FH             ;;mantiene i bit meno significativi
         CURSORE POS                ;;posiziona la cifra sullo schermo
         STAMPANUM                  ;;e la visualizza
         MOV     AL,TEMPO           ;;prepara la seconda cifra
         AND     AL,0F0H            ;;mantiene i bit più significativi
         MOV     CL,04              ;;rotazione alla posizione LSB
         ROR     AL,CL
         CURSORE POS-1              ;;visualizzazione a sinistra di LSB
         STAMPANUM
         POP     CX
         POP     AX
         ENDM

STACK    SEGMENT PARA STACK    ;definisce il segmento di stack
         DB      48 DUP ('STACK ')
STACK    ENDS                       ;fine del segmento di stack

DATI     SEGMENT PARA 'DATI'
SEC12    DB      0
SEC34    DB      0
SEC56    DB      0
DATI     ENDS
```

```
CODICE    SEGMENT PARA 'CODICE' ;definisce il segmento di codice
PROCEDURA PROC    FAR           ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS            ;salva DS sullo stack
          SUB     AX,AX         ;azzera AX
          PUSH    AX            ;salva 0 sullo stack
          MOV     AX,DATI       ;indirizzo di DATI in AX
          MOV     DS,AX         ;indirizzo di DATI in DS
          CANCELLA              ;cancella lo schermo
CICLO:    RT      01            ;chiamata della macro RT (1 sec)
          MOV     AL,SEC12      ;carica il dato
          ADD     AL,1          ;incrementa il conteggio in secondi
          DAA                   ;sistemazione decimale
          MOV     SEC12,AL      ;memorizza i risultati
          MOV     AL,SEC34      ;carica il dato
          ADC     AL,0          ;incrementa, se Carry = 1
          DAA                   ;sistemazione decimale
          MOV     SEC34,AL      ;memorizza i risultati
          MOV     AL,SEC56      ;carica il dato
          ADC     AL,0          ;incrementa, se Carry = 1
          DAA                   ;sistemazione decimale
          MOV     SEC56,AL      ;memorizza i risultati
          SCHERMO SEC12,0C2BH   ;visualizzazione sullo schermo
          SCHERMO SEC34,0C29H
          SCHERMO SEC56,0C27H
          CURSORE 1900H         ;posizionamento cursore
          MOV     AH,06H        ;lettura da tastiera
          MOV     DL,0FFH
          INT     21H
          CMP     AL,'Q'        ;esce, se Q
          JE      FINE
          CMP     AL,'q'        ;esce, se q
          JE      FINE
          JMP     CICLO         ;ripete il ciclo

FINE:

          RET                   ;il controllo ritorna al DOS
PROCEDURA ENDP                  ;fine della procedura
CODICE    END                   ;fine del segmento di codice

          END                   ;fine del programma
```

Figura 8.2 Programma che esegue il conteggio dei secondi

```
          PUSH    BX
          PUSH    CX
          PUSH    DX
          MOV     AH,2CH        ;;lettura ora corrente
          INT     21H
          MOV     BH,DH         ;;somma del ritardo ai secondi correnti
          ADD     BH,ATTESA
          CMP     BH,60         ;;aggiornamento non oltre i 60 secondi
          JL      ANCORA
          SUB     BH,60
ANCORA:   MOV     AH,2CH        ;;rilettura dell'ora
```

```
        INT     21H
        CMP     BH,DH           ;;è finito il ritardo?
        JNE     ANCORA
        POP     DX              ;;ripristina i registri
        POP     CX
        POP     BX
        POP     AX
        ENDM
```

Innanzitutto, questa macro esamina l'ora corrente e la invia al calcolatore, utilizzando il registro DH. La quantità di ritardo richiesta viene sommata all'ora corrente; il risultato viene regolato in modo tale che non superi i 60 secondi e poi viene memorizzato nel registro BH. Successivamente, la macro entra in un ciclo per eseguire ripetutamente il campionamento dell'ora e il confronto con il valore contenuto in BH. Solo quando i due valori coincidono, la macro termina. È possibile realizzare attese a tempo molto accurate in base alle interruzioni hardware disponibili, come ad esempio INT21H. Anche la macro SCHERMO, qui di seguito riportata, presenta alcuni aspetti interessanti:

```
SCHERMO MACRO       TEMPO,POS ;;visualizza le cifre sullo schermo
        PUSH        AX
        PUSH        CX
        MOV         AL,TEMPO  ;;cifra da visualizzare
        AND         AL,0FH    ;;mantiene i bit meno significativi
        CURSORE     POS       ;;posiziona la cifra sullo schermo
        STAMPANUM             ;;e la visualizza
        MOV         AL,TEMPO  ;;prepara la seconda cifra
        AND         AL,0F0H   ;;mantiene i bit più significativi
        MOV         CL,04     ;;rotazione alla posizione LSB
        ROR         AL,CL
        CURSORE     POS-1     ;;visualizzazione a sinistra di LSB
        STAMPANUM
        POP         CX
        POP         AX
        ENDM
```

Questa macro dispone di due argomenti formali, TEMPO e POS. I valori delle tre variabili SEC12, SEC34, SEC56 vengono passati alla macro tramite l'argomento formale TEMPO, mentre POS definisce la posizione del cursore in fase di visualizzazione delle cifre sullo schermo. TEMPO contiene due cifre BCD compattate, la cui visualizzazione avviene separatamente: infatti, inizialmente, i quattro bit più significativi di TEMPO vengono mascherati. Quando la macro di libreria STAMPANUM viene invocata, la cifra meno significativa viene convertita in un numero ASCII e inviata sullo schermo in corrispondenza della posizione corrente del cursore. Successivamente, l'argomento TEMPO viene utilizzato nuovamente, in modo da mascherare i suoi

quattro bit meno significativi. La cifra più significativa, quindi, viene visualizzata sullo schermo in maniera analoga a quanto fatto per la cifra meno significativa. Questa operazione viene ripetuta fino a quando tutte le sei cifre (cioè le tre variabili) appaiono sullo schermo.
La parte di codice responsabile della visualizzazione delle cifre decimali è la seguente:

```
CICLO:  RT       01           ;chiamata della macro RT (1 sec)
        MOV      AL,SEC12     ;carica il dato
        ADD      AL,1         ;incrementa il conteggio in secondi
        DAA                   ;sistemazione decimale
        MOV      SEC12,AL     ;memorizza i risultati
        MOV      AL,SEC34     ;carica il dato
        ADC      AL,0         ;incrementa, se Carry = 1
        DAA                   ;sistemazione decimale
        MOV      SEC34,AL     ;memorizza i risultati
        MOV      AL,SEC56     ;carica il dato
        ADC      AL,0         ;incrementa, se Carry = 1
        DAA                   ;sistemazione decimale
        MOV      SEC56,AL     ;memorizza i risultati
        SCHERMO  SEC12,0C2BH  ;visualizzazione sullo schermo
        SCHERMO  SEC34,0C29H
        SCHERMO  SEC56,0C27H
        CURSORE  1900H        ;posizionamento cursore
```

Questa parte di codice genera innanzitutto un'attesa di un secondo, invocando la macro RT, a cui passa il valore 1. Soddisfatta questa richiesta, il programma aggiorna il valore delle tre variabili che definiscono il conteggio: carica la variabile SEC12 nel registro AL, ne incrementa di una unità il contenuto e converte nel formato decimale il risultato (è indispensabile che il numero sia nel registro AL, per realizzare un conteggio decimale). Il comando DAA deve essere immediatamente seguito dall'istruzione ADD e il risultato viene memorizzato nella variabile SEC12. Se, a seguito di questa somma, il bit Carry contiene il valore 1, è indispensabile incrementare di una unità il contenuto di SEC34 e su questo valore, caricato in AL, viene eseguita l'operazione ADC (invece che ADD). Se il bit Carry è a 1, viene sommato 1 al contenuto del registro AL e il risultato viene salvato dopo che su di esso è stata eseguita l'operazione DAA. Lo stesso procedimento viene seguito per adattare il contenuto della variabile SEC56 alle esigenze del programma. In questo modo, è possibile ottenere l'accuratezza desiderata.
Da ultimo, viene invocata la macro CURSORE 1990H, in modo da spostare il cursore dal punto in cui sono stati visualizzati i numeri. In questo modo, si evita di avere un effetto di luminosità intermittente in corrispondenza di una o più cifre visualizzate.
L'ultima parte del programma verifica se l'utente intende cancellare il conteggio dal video.

```
MOV     AH,06H     ;lettura da tastiera
MOV     DL,0FFH
INT     21H
CMP     AL,'Q'     ;esce, se Q
JE      FINE
CMP     AL,'q'     ;esce, se q
JE      FINE
JMP     CICLO      ;ripete il ciclo
```

Invocando l'interruzione DOS 21H, avviene la lettura di un solo carattere da tastiera e se questo è la lettera Q (maiuscola o minuscola) il programma termina l'esecuzione.
Questo esempio ha illustrato come realizzare una routine di attesa a tempo e come eseguire operazioni decimali in multipla precisione senza ricorrere a complicate routine di conversione da esadecimale a decimale. Inoltre, ha indicato come posizionare un numero sullo schermo e come uscire elegantemente da un ciclo. Potete, a questo punto, con un piccolo sforzo, realizzare un contatore ore-minuti-secondi, inserendo nel programma gli opportuni controlli (come ad esempio CMP SEC12,59).

8.3 Semplice programma a menu

Problema 3: Scrivere un programma interattivo che visualizzi sullo schermo alcune informazioni che riguardano un insieme di persone e che sia semplice da utilizzare per chi non è molto esperto di calcolatori.

I programmi interattivi a menu costituiscono la parte essenziale del software commerciale. Infatti, per rendere più semplice l'uso di un programma – qualunque esso sia – di solito viene visualizzato sullo schermo un insieme di opzioni (menu), che l'utente seleziona in modo che il programma esegua quanto da lui richiesto.
In questo semplice esempio, all'utente viene presentato un menu che elenca alcune informazioni che riguardano un individuo i cui dati sono stati precedentemente memorizzati su file. Le opzioni permettono di selezionare il nome, l'indirizzo, il numero di telefono, il numero di previdenza sociale e la ricetta preferita dal suddetto individuo. Inoltre il menu comprende anche l'opzione per abbandonare il programma. Non viene eseguito il controllo sulla correttezza delle risposte fornite dall'utente, ma il programma ignora qualunque cosa venga inserita da tastiera che non coincida con le lettere elencate nel menu. Il programma (Figura 8.3), inoltre, utilizza la libreria di macro MACLIB.MAC, di cui abbiamo già parlato nel precedente capitolo.
Tutta l'informazione che costituisce il menu è memorizzata in un segmento dati del programma. Anche per un semplice programma, queste informazio-

```
;programma per macchine 8088/80386
;programma che illustra come realizzare un menu di opzioni in
;linguaggio assemblatore

PAGE ,132                           ;dimensione pagina 66 per 132

IF1                                 ;include una libreria di macro
        INCLUDE C:MACLIB.MAC
ENDIF

DISPLAY  MACRO   STRINGA            ;;cancella lo schermo e visualizza
         CANCELLA                   ;;la stringa di caratteri
         CURSORE 0800H              ;;genera un ritardo
         STAMPACHAR STRINGA
         RITARDO 45H
         ENDM

STACK    SEGMENT PARA STACK    ;definisce il segmento di stack
         DB      48 DUP ('STACK ')
STACK    ENDS

DATI     SEGMENT PARA 'DATI'   ;definisce l'inizio di un segmento dati
NOME     DB      '         Paolo Rossi$'

IND      DB            Via Giuseppe Verdi 100',0DH,0AH
                              Milano$'

TEL      DB               02-3475646$'

PS       DB               005-10-1983$'

RIC      DB      'Spaghetti alla amatriciana',0DH,0AH,0AH
         DB      'Per sei persone: 700 grammi di spaghetti - '0DH,0AH
DB'Una cipolla - 100 grammi di guanciale - Un ',0DH,0AH
DB'cucchiaio di strutto - Un chilogrammo di ',0DH,0AH
DB'pomodori - Sale - Pepe - 100 grammi di ',0DH,0AH
DB'pecorino romano.',0DH,0AH,0AH
         DB      'Tagliuzzate sul tagliere la cipolla col ',0DH,0AH
DB'guanciale e mettete il trito in un tegame ',0DH,0AH
DB'con una cucchiaiata di strutto. Quando il ',0DH,0AH
DB'guanciale e la cipolla saranno rosolati ma ',0DH,0AH
DB'non troppo, unite nel tegame i pomodori ',0DH,0AH
DB'spelati, tagliati a pezzi e privati di ',0DH,0AH
DB'semi. Condite con sale e pepe e conducete ',0DH,0AH
DB'la cottura a fuoco brillante, pochi ',0DH,0AH
DB'minuti, fino a che il pomodoro sarà cotto, ',0DH,0AH
DB'ma non sfatto. Nel frattempo buttate giù ',0DH,0AH
'gli spaghetti e, appena arrivati alla ',0DH,0AH
'cottura, conditeli con la salsa preparata ',0DH,0AH
'e il pecorino grattato.',0DH,0AH,0AH
         DB      '$'
MESS     DB      'Indicare la scelta premendo il tasto: ',0DH,0AH,0AH
         DB      'N - Nome',0DH,0AH
         DB      'I - Indirizzo',0DH,0AH
         DB      'T - Telefono',0DH,0AH
         DB      'P - Numero di previdenza sociale',0DH,0AH
         DB      'R - Ricetta preferita',0DH,0AH
         DB      'U - Abbandono del programma',0DH,0AH
         DB      '$'
DATI     ENDS
```

```
CODICE    SEGMENT PARA 'CODICE' ;definisce il segmento di codice
PROCEDURA PROC    FAR           ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS            ;salva DS sullo stack
          SUB     AX,AX         ;azzera AX
          PUSH    AX            ;salva 0 sullo stack
          MOV     AX,DATI       ;indirizzo di DATI in AX
          MOV     DS,AX         ;indirizzo di DATI in DS
RIPETE:   CANCELLA              ;cancella lo schermo
          CURSORE 0200H         ;sposta il cursore
          STAMPACHAR MESS       ;visualizza il messaggio
          MOV     AH,1          ;parametro per tastiera
          INT     21H           ;interruzione per lettura tastiera
          CMP     AL,'N'        ;confronta 'N' con AL
          JNE     NON           ;salta a NON, se non è uguale
          DISPLAY NOME          ;altrimenti, visualizza il nome
NON:      CMP     AL,'I'        ;confronta 'I' con AL
          JNE     NOI           ;salta a NOI, se non è uguale
          DISPLAY IND           ;altrimenti, visualizza l'indirizzo
NOI:      CMP     AL,'T'        ;confronta 'T' con AL
          JNE     NOT           ;salta a NOT, se non è uguale
          DISPLAY TEL           ;altrimenti, visualizza il telefono
NOT:      CMP     AL,'P'        ;confronta 'P' con AL
          JNE     NOP           ;salta a NOP, se non è uguale
          DISPLAY PS            ;altrimenti, visualizza la previdenza
NOP:      CMP     AL,'R'        ;confronta 'R' con AL
          JNE     NOI           ;salta a NOR, se non è uguale
          DISPLAY RIC           ;altrimenti, visualizza la ricetta
NOR:      CMP     AL,'U'        ;confronta 'U' con AL
          JNE     FINE          ;salta a FINE, se non è uguale
          JMP     RIPETE        ;altrimenti, ripete il programma

FINE:

          RET                   ;il controllo ritorna al DOS
PROCEDURA ENDP                  ;fine della procedura
CODICE    END                   ;fine del segmento di codice

          END                   ;fine del programma
```

Figura 8.3 Esempio di programma a menu

ni possono richiedere una grande quantità di codice, per cui per programmi di grandi dimensioni, può essere necessario utilizzare i 64 kB di memoria referenziabili con il registro ES (Extra Segment).
Riportiamo qui di seguito la macro DISPLAY:

```
DISPLAY  MACRO       STRINGA   ;;cancella lo schermo e visualizza
         CANCELLA              ;;la stringa di caratteri
         CURSORE     0800H     ;;genera un ritardo
         STAMPACHAR  STRINGA
         RITARDO     45H
         ENDM
```

Questa macro è responsabile della visualizzazione delle opzioni che l'utente può selezionare dal menu. La stessa macro utilizza quattro altre macro che sono contenute nella libreria MACLIB.MAC. Ogni volta che il programma principale assegna un valore all'argomento formale STRINGA, la macro DISPLAY cancella lo schermo, sposta il cursore alla posizione fissa 0800H, visualizza l'informazione e la lascia sullo schermo per un certo periodo di tempo. Si tenga presente che la macro RITARDO è una routine software che genera un'attesa a tempo e che può essere tranquillamente utilizzata quando l'accuratezza del ritardo non è un fattore critico. Il numero che viene passato alla macro RITARDO è in correlazione indiretta con il tempo, dal momento che rappresenta il numero di volte che la macro deve eseguire un ciclo interno. È semplice, comunque, modificare il valore del numero, per ottenere un tempo di osservazione adatto alle esigenze dell'utenza.
La seguente parte di codice visualizza il menu sullo schermo e attende una risposta da tastiera:

```
RIPETE:  CANCELLA                    ;cancella lo schermo
         CURSORE        0200H        ;sposta il cursore
         STAMPACHAR     MESS         ;visualizza il messaggio
         MOV            AH,1         ;parametro per tastiera
         INT            21H          ;interruzione per lettura tastiera
```

Anche qui vengono invocate alcune macro appartenenti alla libreria MACLIB.MAC. La macro CANCELLA prepara lo schermo alla visualizzazione del menu da parte della macro STAMPACHAR, in corrispondenza della posizione definita dalla macro CURSORE. Gli utenti possono esaminare il menu per tutto il tempo che vogliono, in quanto non viene intrapresa alcuna azione fino a quando non viene premuto un tasto. L'interruzione DOS INT 21H memorizza il valore del tasto premuto nel registro AL. Il resto del programma controlla che il registro AL contenga un valore corretto e poi invoca la macro DISPLAY perché venga visualizzata sullo schermo la risposta all'opzione selezionata.
Scrivete ed eseguite questo programma, che risponderà molto velocemente ai vostri comandi. Tenete presente che una volta che è stata generata la versione eseguibile del programma, non interessa più conoscere il contenuto del file sorgente. La differenza esistente tra un programma in linguaggio assemblatore e un programma scritto in un linguaggio di alto livello consiste solo nella maggiore velocità di esecuzione del primo.

8.4 Programma interattivo a menu di maggiore complessità

Problema 4: Scrivere un programma a menu che chieda all'utente di inserire da tastiera due numeri e poi li moltiplichi o li sommi insieme, visualizzando sullo schermo il risultato dell'operazione.

Innanzitutto è indispensabile precisare come acquisire i numeri da tastiera, in che formato esprimerli, come eseguire le operazioni matematiche e come visualizzare i risultati sullo schermo. Ricorriamo ai programmi presentati precedentemente. L'ultimo programma ha indicato un metodo per visualizzare un menu sullo schermo e per leggere i caratteri da tastiera. Il programma di Figura 8.2 ha mostrato come i numeri possono essere visualizzati sullo schermo in posizioni predefinite del cursore.
Per limitare la complessità di questo problema, ipotizziamo che l'utente possa inserire da tastiera solo numeri decimali a singola cifra e che le operazioni matematiche realizzate dal programma si limitino alla somma e alla moltiplicazione. Questo significa che il massimo valore ottenibile con la somma è 18 (9 + 9) e con la moltiplicazione è 81 (9 × 9).
L'istruzione di chiamata dell'interruzione DOS 21H genera la richiesta di un carattere da tastiera e restituisce nel registro AL l'equivalente ASCII del tasto premuto. I numeri a singola cifra possiedono un campo di valori ASCII compreso tra 30H e 39H, e con questo formato vengono visualizzati sullo schermo. Per eseguire le operazioni aritmetiche, però, è indispensabile convertire i numeri nel formato decimale, sottraendo alla loro codifica ASCII il valore 30H.
Esaminiamo il programma di Figura 8.4: si nota immediatamente che vengono utilizzate due macro, in aggiunta a quelle contenute nella libreria, e una procedura di tipo NEAR per visualizzare i risultati sullo schermo. Poiché le routine possono essere codificate come macro o come procedure, la scelta di definire ARITM e TASTO come macro e PRINTRIS come procedura è dovuta alle seguenti considerazioni: TASTO contiene solo cinque linee di codice e viene chiamata solo tre volte dal programma; è naturale codificare TASTO come macro. ARITM e PRINTRIS hanno circa la stessa lunghezza, ma ARITM viene invocata solo una volta dal programma, per cui entrambe le soluzioni vanno bene (il suo codice non viene espanso ripetutamente nel programma). La scelta di una macro per codificare ARITM permette di disporre di codice in sequenza e di ottenere una velocità di esecuzione maggiore. Infine, è naturale che PRINTRIS sia una procedura NEAR, in quanto il codice è abbastanza lungo e, soprattutto, viene invocata tre volte dal programma. In questo modo viene ottimizzata la dimensione del programma.
Esaminiamo ognuna delle tre routine (ARITM, TASTO e PRINTRIS) prima di procedere alla spiegazione del programma principale.

```
;per macchine 80286/80386
;programma interattivo che visualizza un menu di opzioni sullo
;schermo, attraverso cui l'utente sceglie se eseguire la somma o
;la moltiplicazione di due numeri a singola cifra inseriti da
;tastiera.

PAGE ,132                       ;dimensione pagina 66 per 132

IF1                             ;include una libreria di macro
        INCLUDE C:MACLIB.MAC
ENDIF

ARITM   MACRO                   ;;esegue operazioni aritmetiche
        PUSH    AX              ;;salva i registri soggetti a modifiche
        PUSH    CX
        SUB     AX,AX           ;;azzera AX
        MOV     AL,NUM1         ;;primo numero in AL
        CMP     OPER,'S'        ;;analisi del tipo di operazione voluta
        JE      SOMMA           ;;se è la somma, salta a SOMMA
MOLT:   MUL     NUM2            ;;altrimenti, moltiplica
        AAM                     ;;sistemazione ASCII del risultato
        JMP     FINE            ;;salta a FINE
SOMMA:  ADD     AL,NUM2         ;;esegue la somma
        AAA                     ;;sistemazione ASCII del risultato
FINE:   AND     AH,OFH          ;;mantiene solo i quattro bit meno significativi di AH
        MOV     CL,04H          ;;numero di spostamenti
        SHL     AH,CL           ;;traslazione
        ADD     AL,AH           ;;risultato finale in AL
        MOV     RIS,AL          ;;risultato finale in RIS
        POP     CX              ;;ripristino dei registri modificati
        POP     AX
        ENDM

TASTO   MACRO   QUESTO          ;;lettura di un tasto
        PUSH    AX              ;;salva il registro soggetto a modifiche
        MOV     AH,01H          ;;parametro per tastiera
        INT     21H             ;;lettura di un tasto
        MOV     QUESTO,AL       ;;codice tasto in QUESTO
        POP     AX              ;;ripristino del registro modificato
        ENDM

STACK   SEGMENT PARA STACK      ;definisce il segmento di stack
        DB      48 DUP ('STACK ')
STACK   ENDS                    ;fine del segmento di stack

DATI    SEGMENT PARA 'DATI'
PIU     DB      '+$'
PER     DB      'x$'
UGUALE  DB      '=$'
MENU1   DB      '   SCEGLI L''OPERAZIONE SELEZIONANDO UNA DELLE SEGUENTI TRE OPZIONI:'
        DB      '                                                                     '
        DB      '                          A - SOMMA DUE NUMERI                       '
        DB      '                                                                     '
        DB      '                        M - MOLTIPLICA DUE NUMERI                    '
        DB      '                                                                     '
        DB      '                         E - FINE DEL PROGRAMMA                      '
        DB      '                                                                     '
        DB      '$'
MENU2   DB      'Inserisci il primo numero (0-9):       $'
MENU3   DB      'Inserisci il secondo numero (0-9):     $'
```

```
VALORE    DB      ?
POSI      DB      ?
OPER      DB      ?
NUM1      DB      0
NUM2      DB      0
RIS       DB      0
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE' ;definisce il segmento di codice
PROCEDURA PROC    FAR           ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS            ;salva DS sullo stack
          SUB     AX,AX         ;azzera AX
          PUSH    AX            ;salva 0 sullo stack
          MOV     AX,DATI       ;indirizzo di DATI in AX
          MOV     DS,AX         ;indirizzo di DATI in DS

;codice che visualizza il menu principale sullo schermo
MMENU:    CANCELLA              ;cancella lo schermo
          CURSORE 0C00H         ;posizionamento cursore
          STAMPACHAR MENU1      ;visualizza il menu principale

;codice che controlla se deve avere luogo una operazione
SCELTA:   TASTO   OPER          ;legge il comando da tastiera
          CMP     OPER,'E'      ;fine del programma?
          JNE     NUMERO        ;se no, legge il primo numero
          JMP     STOP          ;se si, stop
NUMERO:   CANCELLA              ;cancella il menu
          CURSORE 0C00H         ;posizionamento del cursore
          STAMPACHAR MENU2      ;richiesta del primo numero

;lettura del primo numero a singola cifra
          TASTO   NUM1          ;legge il primo numero
          SUB     NUM1,30H      ;conversione decimale
          CURSORE 0C00H         ;posizionamento del cursore
          CANCELLA              ;cancella lo schermo
          STAMPACHAR MENU3      ;visualizza il terzo menu

;lettura del secondo numero a singola cifra
          TASTO   NUM2          ;legge il secondo numero
          SUB     NUM2,30H      ;conversione decimale
          CANCELLA              ;cancella lo schermo

;realizzazione dell'operazione aritmetica
          ARITM                 ;operazione di somma o di moltiplicazione
          MOV     AL,NUM1       ;preparativi per visualizzare NUM1
          MOV     VALORE,AL
          MOV     POS,0C1FH
          CALL    PRINTRIS
          CURSORE 0C22H
          CMP     OPER,'S'
          JNE     P
          STAMPACHAR PIU        ;visualizza + nel caso di somma
          JMP     ANCORA
P:        STAMPACHAR PER        ;visualizza x nel caso di moltiplicazione
ANCORA:   MOV     AL,NUM2       ;preparativi per visualizzare NUM2
          MOV     VALORE,AL
          MOV     POS,0C24H
          CALL    PRINTRIS
          CURSORE 0C27H
          STAMPACHAR UGUALE     ;visualizza il segno =
```

```
        MOV     AL,RIS          ;preparativi per visualizzare RIS
        MOV     VALORE,AL
        MOV     POS,0C29H
        CALL    PRINTRIS
        RITARDO 25H             ;tempo di attesa per la lettura del risultato
        JMP     MMENU           ;ritorna al menu principale
STOP:
        RET                     ;il controllo ritorna al DOS
PROCEDURA ENDP                  ;fine di PROCEDURA

PRINTRIS PROC   NEAR            ;procedura per visualizzare i numeri
        PUSH    AX              ;salva i registri soggetti a modifiche
        PUSH    CX
        PUSH    DX
        MOV     AL,VALORE       ;carica il dato
        AND     AL,0F0H         ;mantiene solo i quattro bit più significativi
        MOV     CL,4            ;numero di rotazioni
        ROR     AL,CL           ;rotazione
        CURSORE POS             ;chiamata della macro CURSORE
        STAMPANUM               ;macro STAMPANUM
        MOV     AL,VALORE       ;carica il dato
        AND     AL,0FH          ;mantiene solo i quattro bit meno significativi
        ADD     POS,01H         ;visualizza a sinistra la prima cifra
        CURSORE POS             ;chiamata della macro CURSORE
        STAMPANUM               ;macro STAMPANUM
        POP     DX              ;ripristino dei registri modificati
        POP     CX
        POP     AX
        RET                     ;fine di PRINTRIS
PRINTRIS ENDP

CODICE  ENDS                    ;fine del segmento di codice

        END                     ;fine del programma
```

Figura 8.4 Programma che esegue la somma o la moltiplicazione di due numeri a singola cifra, inseriti da tastiera, in base alla scelta delle opzioni di menu operata dall'utente

Il codice di ARITM è il seguente:

```
ARITM   MACRO               ;;esegue operazioni aritmetiche
        PUSH    AX          ;;salva i registri soggetti a modifiche
        PUSH    CX
        SUB     AX,AX       ;;azzera AX
        MOV     AL,NUM1     ;;primo numero in AL
        CMP     OPER,'S'    ;;analisi del tipo di operazione voluta
        JE      SOMMA       ;;se è la somma, salta a SOMMA
MOLT:   MUL     NUM2        ;;altrimenti, moltiplica
        AAM                 ;;sistemazione ASCII del risultato
        JMP     FINE        ;;salta a FINE
SOMMA:  ADD     AL,NUM2     ;;esegue la somma
        AAA                 ;;sistemazione ASCII del risultato
FINE:   AND     AH,0FH      ;;mantiene solo i quattro bit meno significativi
                            ;;di AH
```

```
         MOV     CL,04H          ;;numero di spostamenti
         SHL     AH,CL           ;;traslazione
         ADD     AL,AH           ;;risultato finale in AL
         MOV     RIS,AL          ;;risultato finale in RIS
         POP     CX              ;;ripristino dei registri modificati
         POP     AX
         ENDM
```

Il segmento dati contiene anche due variabili (NUM1 e NUM2), che sono state definite con la direttiva DB (define byte) e che memorizzano i numeri a singola cifra inseriti da tastiera dall'utente. Inoltre, l'utente può selezionare il tipo di operazione che il programma deve realizzare ('S' per la somma e 'M' per la moltiplicazione) e questa opzione viene memorizzata nella variabile OPER. NUM1 viene trasferito nel registro AL e poi, in base al tipo di operazione richiesta, viene eseguito il codice referenziato dalla etichetta SOMMA o dalla etichetta MOLT. Si tenga presente che, in entrambi i casi, le operazioni aritmetiche vengono eseguite su numeri decimali compattati. Questo comporta l'uso dell'istruzione AAM, dopo l'operazione di moltiplicazione MUL, e l'uso dell'istruzione AAA, dopo l'operazione di somma ADD, per effettuare la sistemazione ASCII dei risultati (si consulti a questo proposito) il Capitolo 3). Prima di salvare i risultati nella variabile RIS, le due cifre decimali vengono isolate e compattate nel registro AL.
Riportiamo qui di seguito la macro TASTO:

```
TASTO    MACRO      QUESTO       ;;lettura di un tasto
         PUSH       AX           ;;salva il registro soggetto a modifiche
         MOV        AH,01H       ;;parametro per tastiera
         INT        21H          ;;lettura di un tasto
         MOV        QUESTO,AL    ;;codice tasto in QUESTO
         POP        AX           ;;ripristino del registro modificato
         ENDM
```

TASTO utilizza l'argomento formale QUESTO per restituire la codifica ASCII del tasto premuto. Si tenga presente che, quando la macro viene invocata, QUESTO assume il valore contenuto in una delle tre variabili OPER, NUM1 e NUM2.
Il codice della procedura PRINTRIS è simile al codice scritto per visualizzare sullo schermo il conteggio in secondi (Figura 8.2):

```
PRINTRIS PROC       NEAR         ;procedura per visualizzare i numeri
         PUSH       AX           ;salva i registri soggetti a modifiche
         PUSH       CX
         PUSH       DX
         MOV        AL,VALORE    ;carica il dato
         AND        AL,0F0H      ;mantiene solo i quattro bit più
                                 ;significativi
         MOV        CL,4         ;numero di rotazioni
```

```
          ROR          AL,CL        ;rotazione
          CURSORE      POS          ;chiamata della macro CURSORE
          STAMPANUM                 ;macro STAMPANUM
          MOV          AL,VALORE    ;carica il dato
          AND          AL,0FH       ;mantiene solo i quattro bit meno
                                    ;significativi
          ADD          POS,01H      ;visualizza a sinistra la prima cifra
          CURSORE      POS          ;chiamata della macro CURSORE
          STAMPANUM                 ;macro STAMPANUM
          POP          DX           ;ripristino dei registri modificati
          POP          CX
          POP          AX
          RET                       ;fine di PRINTRIS
PRINTRIS  ENDP
```

Questa procedura è responsabile della visualizzazione dei due numeri a singola cifra inseriti da tastiera e del risultato della loro elaborazione. Poiché le procedure in linguaggio assemblatore non permettono l'uso di parametri formali, il numero da visualizzare deve essere passato alla procedura stessa tramite la variabile VALORE. PRINTRIS visualizza la cifra più significativa del risultato e, dopo aver aggiornato la posizione del cursore, ne visualizza anche la cifra meno significativa. Vengono invocate due macro appartenenti alla libreria MACLIB.MAC: CURSORE, che utilizza la variabile POS per stabilire in quale posizione dello schermo visualizzare i numeri, e STAMPANUM, che converte la cifra decimale compattata nella codifica ASCII, in modo da rendere possibile la seguente visualizzazione sullo schermo.
I numeri sono stati memorizzati non compattati, poi compattati, e di nuovo non compattati, per le seguenti ragioni: le istruzioni di somma e moltiplicazione richiedono i numeri nel formato non compattato; le istruzioni AAA e AAM memorizzano i risultati nei registri in formati leggermente differenti: AAA memorizza la risposta in AL, mentre AAM la memorizza in AH e AL. Questi risultati vengono compattati in quanto esiste una routine (PRINTRIS) che visualizza sullo schermo numeri BCD a doppia cifra compattati.
In definitiva, il programma, quando viene eseguito, visualizza sullo schermo il menu principale che è memorizzato nel segmento dati. A questo livello, l'utente può già decidere se sommare o moltiplicare due numeri, oppure se uscire dal programma.

```
;codice che visualizza il menu principale sullo schermo
MMENU:   CANCELLA                       ;cancella lo schermo
         CURSORE        0C00H           ;posizionamento cursore
         STAMPACHAR     MENU1           ;visualizza il menu principale
;codice che controlla se deve avere luogo una operazione
SCELTA:  TASTO          OPER            ;legge il comando da tastiera
         CMP            OPER,'E'        ;fine del programma?
         JNE            NUMERO          ;se no, legge il primo numero
```

```
          JMP          STOP        ;se sì, stop
NUMERO:   CANCELLA                 ;cancella il menu
          CURSORE      0C00H       ;posizionamento del cursore
          STAMPACHAR   MENU2       ;richiesta del primo numero
```

Le prime tre linee di codice permettono di cancellare lo schermo, aggiornano la posizione del cursore e visualizzano il menu principale. Le successive istruzioni richiedono all'utente di specificare un comando da tastiera: se questo non corrisponde alla lettera E (esci), il programma prosegue e richiede all'utente di inserire un numero.

```
;lettura del primo numero a singola cifra
          TASTO        NUM1        ;legge il primo numero
          SUB          NUM1,30H    ;conversione decimale
          CURSORE      0C00H       ;posizionamento del cursore
          CANCELLA                 ;cancella lo schermo
```

Analogamente, il programma richiede all'utente un secondo numero:

```
;lettura del secondo numero a singola cifra
          TASTO        NUM2        ;legge il secondo numero
          SUB          NUM2,30H    ;conversione decimale
          CANCELLA                 ;cancella lo schermo
```

Dopo che l'utente ha inserito i due numeri da tastiera, il programma esegue su di essi l'operazione aritmetica e visualizza il risultato sullo schermo:

```
;realizzazione dell'operazione aritmetica
          ARITM        ;operazione di somma o di moltiplicazione
```

Il programma principale invoca la macro ARITM, che esegue l'operazione aritmetica selezionata memorizzando il risultato a doppia cifra nella variabile RIS con il formato BCD compattato:

```
          MOV     AL,NUM1      ;preparativi per visualizzare NUM1
          MOV     VALORE,AL
          MOV     POS,0C1FH
          CALL    PRINTRIS
```

La precedente parte di programma prepara il contenuto di NUM1 per la procedura PRINTRIS, aggiorna la posizione del cursore al valore 0C1FH, ed invoca la procedura stessa perché visualizzi il numero a sinistra del centro dello schermo. Spostando il cursore a destra, viene ora visualizzato il simbolo dell'operazione che è stata eseguita dal programma (somma o moltiplicazione):

```
     CURSORE      0C22H
     CMP          OPER,'S'
```

```
        JNE           P
        STAMPACHAR    PIU          ;visualizza + nel caso di somma
        JMP           ANCORA
P:      STAMPACHAR    PER          ;visualizza x nel caso di moltiplicazione
```

I simboli di somma (+) e di moltiplicazione (×) erano stati memorizzati nel segmento dati come caratteri, rispettivamente sotto i nomi di variabile PIU e PER, per cui è lecito utilizzare la macro STAMPACHAR per visualizzarli sullo schermo in corrispondenza della posizione corrente del cursore.

```
ANCORA:     MOV     AL,NUM2      ;preparativi per visualizzare NUM2
            MOV     VALORE,AL
            MOV     POS,0C24H
            CALL    PRINTRIS
```

Il secondo numero inserito da tastiera viene visualizzato sullo schermo dalla procedura PRINTRIS, mentre per la visualizzazione del segno di uguaglianza (=) memorizzato come carattere viene invocata ancora la macro STAMPACHAR:

```
            CURSORE       0C27H
            STAMPACHAR    UGUALE   ;visualizza il segno =
```

Infine, il risultato dell'operazione viene visualizzato a destra del segno di uguaglianza:

```
            MOV       AL,RIS       ;preparativi per visualizzare RIS
            MOV       VALORE,AL
            MOV       POS,0C29H
            CALL      PRINTRIS
```

e rimane sullo schermo per un periodo di tempo predefinito, prima che compaia di nuovo il menu principale.

```
            RITARDO   25H          ;tempo di attesa per la lettura del risultato
            JMP       MMENU        ;ritorna al menu principale
```

In questo programma, abbiamo utilizzato una routine di attesa a tempo software in quanto il tempo non è un fattore critico.
L'esempio appena discusso può essere ampliato per realizzare programmi più complessi, in grado ad esempio di eseguire le quattro operazioni aritmetiche (somma, moltiplicazione, sottrazione e divisione), fornendo risultati interi, o, con maggiori modifiche, in grado di eseguire le operazioni anche su numeri a doppia cifra (si esamini il programma sull'onda quadra che viene presentato nel Capitolo 9). In quest'ultimo caso, qualche difficoltà si incontra per scrivere la routine di lettura dei numeri a doppia cifra, ma ben maggiori sono le difficoltà che sorgono per realizzare le operazioni aritmetiche

su questi numeri e per visualizzare i risultati delle elaborazioni sullo schermo. Il programma può anche essere modificato per consentire all'utente di rispondere da tastiera ad un quesito matematico. Questa risposta può essere poi confrontata con quella del calcolatore, in modo da visualizzare sullo schermo l'esito del confronto.

8.5 Comandi di manipolazione di stringhe

Problema 5: Scrivere un programma a menu che permetta all'utente di controllare se la stringa di caratteri inserita da tastiera corrisponde o meno ad una parola di senso comune.

Questo programma è completamente diverso dai precedenti programmi a menu. La differenza non consiste nella definizione del menu, ma nel modo in cui i dati vengono inseriti da tastiera e poi utilizzati all'interno del programma.
I problemi di programmazione da risolvere sono parecchi: come inserire da tastiera le parole e come memorizzarle; come regolarsi con le stringhe di caratteri eccessivamente lunghe; dove memorizzare il dizionario di parole necessario per confrontare la stringa di caratteri inserita dall'utente. Potete scegliere diverse soluzioni per risolvere questi problemi.
Il programma che presentiamo è un semplice esempio di analizzatore di parole. Viene realizzata una ricerca lineare nel dizionario fino a quando la parola in esame viene trovata oppure terminano le parole del dizionario. Nel nostro caso, il dizionario si compone di otto parole che iniziano con la lettera P (potete ampliare il dizionario, se lo desiderate). La Figura 8.5 illustra il programma completo. Riportiamo qui di seguito il contenuto del segmento dati:

```
DATI        SEGMENT  PARA 'DATI'    ;area dati del programma
NOME1       DB       9, 9 DUP (' '),'$'
RICHIESTA   DB       'Inserite la parola da esaminare',0DH,0AH,'$'
MESS1       DB       0DH,0AH,'NON SI TROVA NEL DIZIONARIO$'
MESS2       DB       0DH,0AH,'SI TROVA NEL DIZIONARIO$'
DIZ         DB       'PERICOLO','PIETRA  ','POLIGONO'
            DB       'POLSO   ','POPOLO  ','POSTA   '
            DB       'PREGIO  ','PRETESA '
DATI        ENDS
```

Alla variabile NOME1 viene riservato spazio sufficiente per memorizzare la parola che l'utente inserisce da tastiera. Ognuna delle parole del dizionario ha una dimensione massima di otto caratteri (Questo numero può essere ampliato, per avere dizionari più completi). Il programma, al termine dell'ese-

```
;per macchine 8088/80386
;programma che realizza un semplice analizzatore di parole,
;utilizzando avanzati comandi di stringa
;
PAGE ,132                         ;dimensione pagina 66 per 132

IF1                               ;include una libreria di macro
        INCLUDE C:MACLIB.MAC
ENDIF

DATI      SEGMENT PARA 'DATI'   ;area dati del programma
NOME1     DB      9,9 DUP (' '),'$'
RICHIESTA DB      'Inserite la parola da esaminare',0DH,0AH,'$'
MESS1     DB      0DH,0AH,'NON SI TROVA NEL DIZIONARIO$'
MESS2     DB      0DH,0AH,'SI TROVA NEL DIZIONARIO$'
DIZ       DB      'PERICOLO','PIETRA  ','POLIGONO'
          DB      'POLSO   ','POPOLO  ','POSTA   '
          DB      'PREGIO  ','PRETESA '
DATI      ENDS

STACK     SEGMENT PARA STACK   ;segmento di stack del programma
          DB      48 DUP ('STACK ')
STACK     ENDS

CODICE    SEGMENT PARA 'CODICE' ;definisce il segmento di codice
PROCEDURA PROC    FAR          ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS           ;salva DS sullo stack
          SUB     AX,AX        ;azzera AX
          PUSH    AX           ;salva 0 sullo stack
          MOV     AX,DATI      ;indirizzo di DATI in AX
          MOV     DS,DATI      ;indirizzo di DATI in DS

;menu principale
          CANCELLA             ;cancella lo schermo
          STAMPACHAR RICHIESTA ;visualizza il menu principale

;codice che legge e memorizza la parola inserita da tastiera
          LEA     DX,NOME1     ;dove memorizzare la stringa
          MOV     AH,0AH       ;parametro per lettura da tastiera
          INT     21H          ;lettura da tastiera

;codice che sostituisce il carattere di ritorno  a capo con il
;carattere spazio nelle parole con meno di otto caratteri
          MOV     AL,NOME1+1   ;numero di caratteri massimo
          ADD     AL,2         ;aggiunge 2
          CBW                  ;converte in una word
          MOV     BX,AX        ;trasferisce in BX
          MOV     NOME1[BX],' ' ;sostituisce CR con ' '

;confronto della parola inserita da tastiera con le parole del
;dizionario
          CLD                  ;inizializza il flag direzione
          MOV     BX,00H       ;BX contiene il valore dell'indice
          MOV     AX,00H       ;contatore di parole a zero
NEXTWORD: MOV     CL,0AH       ;dimensione di ogni parola in byte
          LEA     DI,NOME1+2   ;indirizzo effettivo del dato in DI
          LEA     SI,DIZ[BX]   ;indirizzo effettivo della parola corrente in SI
          REPE    CMPSB        ;confronto di 10 byte
          JE      OK           ;tutti i 10 byte delle due parole coincidono?
```

```
        ADD     BX,0AH          ;se no, passa alla prossima parola
        ADD     AX,01H          ;incrementa il contatore di parole
        CMP     AX,08H          ;esaminate tutte le parole disponibili?
        JNE     NEXTWORD        ;se no, continua

;visualizzazione dell'esito del confronto
        STAMPACHAR MESS1        ;la parola non si trova nel dizionario
        JMP     OUT
OK:     STAMPACHAR MESS2        ;la parola si trova nel dizionario
OUT:

        RET                     ;il controllo ritorna al DOS
PROCEDURA ENDP                  ;fine della procedura
CODICE  ENDS                    ;fine del segmento di codice

        END                     ;fine del programma
```

Figura 8.5 Esempio di analizzatore di parole che utilizza avanzati comandi di stringa

cuzione, visualizza sullo schermo un messaggio a seconda dell'esito del confronto (NON SI TROVA NEL DIZIONARIO oppure SI TROVA NEL DIZIONARIO). È possibile che la parola inserita da tastiera abbia un senso comune, ma che non faccia parte del dizionario definito nel programma. Il menu visualizzato sullo schermo si compone di un solo messaggio: 'Inserite la parola da esaminare'.

Il programma si compone di tre parti principali: lettura dei dati in ingresso, confronto di parole e visualizzazione del messaggio sullo schermo. La parte di programma che accetta i dati da tastiera ricorda la soluzione proposta nei programmi presentati nel Capitolo 5:

```
;codice che legge e memorizza la parola inserita da tastiera
        LEA     DX,NOME1        ;dove memorizzare la stringa
        MOV     AH,0AH          ;parametro per lettura da tastiera
        INT     21H             ;lettura da tastiera
```

L'interruzione DOS INT 21H memorizza nella variabile NOME1 la stringa inserita da tastiera, la cui lunghezza dipende dalla quantità di memoria che è stata riservata a NOME1: nel nostro caso, questa quantità corrisponde a nove caratteri, in modo da poter trattare parole di otto lettere (l'istruzione INT 21H – con il registro AH inizializzato a 0AH – restituisce il numero di caratteri letti). È indispensabile anche prevedere e risolvere la situazione di stringhe con un numero di caratteri inferiore a quello massimo, che terminano con un ritorno a capo (CR). Poiché le parole contenute nel dizionario non comprendono il carattere di ritorno a capo, questo carattere deve essere sostituito con il carattere spazio nelle stringhe che l'utente inserisce da tastiera. Questa operazione viene realizzata nel modo seguente:

```
;codice che sostituisce il carattere di ritorno a capo con il
;carattere spazio nelle parole con meno di otto caratteri
```

```
        MOV     AL,NOME1+1      ;numero di caratteri massimo
        ADD     AL,2            ;aggiunge 2
        CBW                     ;converte in una word
        MOV     BX,AX           ;trasferisce in BX
        MOV     NOME1[BX],' '   ;sostituisce CR con ' '
```

La definizione della variabile NOME1 contiene due valori numerici '9': il primo indica il massimo numero di caratteri che possono essere memorizzati in NOME1, mentre il secondo permette di allocare nove spazi bianchi, seguiti dal tappo $ che costituisce il marcatore di fine stringa.
Per stabilire il numero di caratteri che l'utente ha inserito da tastiera, è sufficiente referenziare NOME1 + 1. Per posizionarsi sul carattere di ritorno a capo è indispensabile sommare al valore precedente un offset pari a 2, in quanto il programma deve tenere conto anche del primo '9'. Il registro BX viene utilizzato come indice per spaziare all'interno di NOME1, per cui il valore contenuto in AL viene convertito in una word tramite l'istruzione CBW. Da ultimo, il carattere di ritorno a capo viene sostituito da un carattere spazio.
Il confronto tra la parola inserita da tastiera e le parole contenute nel dizionario viene eseguito dalle seguenti istruzioni:

```
;confronto della parola inserita da tastiera con le parole del dizionario
            CLD                     ;inizializza il flag direzione
            MOV     BX,00H          ;BX contiene il valore dell'indice
            MOV     AX,00H          ;contatore di parole a zero
NEXTWORD:   MOV     CL,0AH          ;dimensione di ogni parola in byte
            LEA     DI,NOME1+2      ;indirizzo effettivo del dato in DI
            LEA     SI,DIZ[BX]      ;indirizzo effettivo della parola
                                    ;corrente in SI
            REPE    CMPSB           ;confronto di 10 byte
            JE      OK              ;tutti i 10 byte delle due parole
                                    ;coincidono?
            ADD     BX,0AH          ;se no, passa alla prossima parola
            ADD     AX,01H          ;incrementa il contatore di parole
            CMP     AX,08H          ;esaminate tutte le parole
                                    ;disponibili?
            JNE     NEXTWORD        ;se no, continua
```

La tecnica utilizzata per eseguire il confronto è quella di ricorrere all'istruzione CMPSB. Se l'esito del confronto è negativo, il programma esamina la successiva parola del dizionario, fino a quando la ricerca fornisce un risultato positivo o non ci sono più parole disponibili da confrontare.
Il registro BX contiene il puntatore alla parola corrente del dizionario, per cui, per posizionarsi all'inizio della parola successiva, è indispensabile sommare a BX il valore numerico corrispondente alla lunghezza della parola puntata. Si tenga presente che tutte le parole hanno una lunghezza di otto caratteri, considerando anche i caratteri spazio. Il registro AX contiene il

numero delle parole che sono state già confrontate con la parola inserita da tastiera e permette di stabilire se continuare o meno il confronto (il numero massimo di confronti è otto). CL memorizza il numero di byte (otto, in questo caso) che devono essere confrontati per ogni parola del dizionario. L'indirizzo effettivo del dato è NOME1+2, in modo che il programma possa saltare le indicazioni relative al numero massimo di lettere per parola e al numero effettivo di lettere di cui si compone la parola. Utilizzando l'istruzione REPE CMPSB, il programma esce immediatamente dal ciclo, non appena il confronto dà esito positivo, e visualizza sullo schermo il messaggio che indica la presenza nel dizionario della parola inserita da tastiera. Se, invece, l'esito del confronto è negativo, il registro BX viene incrementato di 10 (0AH) e il confronto viene eseguito sulla successiva parola del dizionario; se non esistono più parole da confrontare, il programma visualizza sullo schermo il messaggio che indica l'assenza nel dizionario della parola inserita da tastiera.
La soluzione adottata in questo programma per eseguire il confronto tra le parole evidenzia due aspetti negativi: poiché non esiste un algoritmo efficiente di ricerca, anche se la parola inserita da tastiera non è di senso comune o non appartiene al dizionario, oppure, se vi appartiene, coincide con l'ultima parola memorizzata, viene ugualmente eseguito il confronto con tutte le parole del dizionario. Inoltre, ad ogni elemento del dizionario viene riservata la stessa quantità di memoria, ma risulterebbe più efficiente eseguire un numero di confronti pari al numero di caratteri di cui si compone la parola in ingresso, piuttosto che eseguire, per ogni parola, il numero massimo di confronti. Quest'ultima soluzione però non è applicabile, se non si apportano alcune modifiche al programma, in quanto non si sarebbe in grado di distinguere una parola da un suo prefisso

8.6 Creazione e utilizzo di file su disco

I precedenti cinque programmi hanno evidenziato come risolvere alcuni problemi, utilizzando gli strumenti discussi nei Capitoli 5, 6 e 7. I restanti programmi di questo capitolo enfatizzano concetti di maggiore complessità, per applicare i quali non sono stati sviluppati ancora adeguati strumenti.
I prossimi tre programmi eseguono la gestione di file, un problema questo che viene discusso sommariamente da altri libri che introducono alla programmazione in linguaggio assemblatore. Noi, invece, esamineremo in dettaglio come avviene la lettura e la scrittura su file, essendo questa un'esigenza fondamentale in qualunque linguaggio di programmazione.
La versione DOS 2.0 ha segnato una svolta per quanto riguarda la gestione di file da parte di programmi scritti nel linguaggio assemblatore. È vero che la gestione dei file può essere implementata a livello BIOS, ma questo non

è sicuramente il metodo più efficiente. Le interruzioni DOS offrono servizi più completi e prestazioni migliori delle routine BIOS. Le versioni DOS 1.0 e 1.1 richiedono l'uso dell'FCB (File Control Block) per accedere ai file mediante la chiamata di interruzione INT 21H (AH=0FH fino a 29H). L'FCB contiene le informazioni sul file: numero del drive, nome del file, estensione del nome del file, numero di blocco corrente, dimensione del record, dimensione del file, data, numero di record e numero del record corrente. Per avere informazioni dettagliate sulla struttura dell'FCB, consultate i manuali DOS fino alla versione DOS 3.0. A partire da questa versione, infatti, l'informazione relativa all'FCB è stata trasferita in un manuale tecnico DOS separato. L'FCB ha permesso al programmatore di avere pieno controllo sui file, ma rappresenta una soluzione non ottima, in quanto deve essere applicata ad ogni accesso ai file. Le versioni correnti di DOS supportano ancora la soluzione a chiamate di interruzioni, ma non si consiglia il loro uso.
I miglioramenti realizzati nella versione DOS 2.0 e ampliati nella versione DOS 3.0 hanno permesso al programmatore di accedere ai file senza ricorrerere all'FCB. Le nuove funzionalità DOS utilizzano ancora l'interruzione INT 21H per accedere ai file, ma le specifiche sono AH=39H fino a 46H. Consultate la tabella di Figura 5.2 per avere maggiori dettagli sui questi comandi. Quando viene creato un file con questi nuovi comandi, il DOS definisce un descrittore per quel file, che viene restituito (registro AX) al programmatore ogni volta che il file viene aperto alla lettura o alla scrittura (analizzate la macro APERTURA nella Figura 8.7). Il descrittore di file esegue automaticamente tutte quelle funzioni che una volta venivano realizzate dall'FCB, ma con minore sforzo da parte del programmatore. Naturalmente, il prezzo pagato è la perdita di controllo sull'FCB.
Altre funzioni DOS memorizzano nel registro AX i codici di errore quando non è lecito aprire, leggere, scrivere o chiudere un file. Il numero iniziale di messaggi di errore (18) presenti nella versione DOS 2.0 è stato ampliato a 39 a partire dalla versione DOS 3.0. Ad esempio, AX=1 indica un numero di funzione non valido, AX=2 indica che il file non è stato trovato, mentre AX=3 specifica un errore di tipo generale. Per avere maggiori dettagli sul significato di questi numeri, consultate il DOS Technical Manual, in modo da conoscere sotto quali condizioni vengono generati i messaggi di errore.
I prossimi tre problemi di programmazione trattano diversi aspetti della gestione di file. Il problema 6 impone di creare un file su dischetto; il problema 7 richiede al programmatore di aprire, scrivere e chiudere un file, mentre il problema 8 riguarda la lettura di un file.

Problema 6: Creazione di un file su disco.

Prima di utilizzare un file per la prima volta, è indipensabile averlo creato (chiamata di interruzione INT 21H con AH=3CH). La Figura 8.6 indica co-

```
;per macchine 8088/80386
;programma che illustra come creare un file (che risulta cosi
;disponibile alla lettura e alla scrittura), utilizzando i
;comandi DOS.

PAGE    ,132                        ;dimensione delle pagine 66 per 132

CREA       MACRO                    ;;macro per creare un file
           MOV      AH,3CH          ;;parametro di INT 21H per creare un file
           MOV      CX,00H          ;;attributo di file normale
           LEA      DX,NOMEFILE     ;;disco/nome/estensione del file da creare
           INT      21H             ;;crea il file
           ENDM

STACK      SEGMENT PARA STACK   ;segmento di stack del programma
           DB       48 DUP ('STACK ')
STACK      ENDS

DATI       SEGMENT PARA 'DATI'
NOMEFILE   DB       'B:NUMERI.DAT',0  ;nome del file da creare
DATI       ENDS

CODICE     SEGMENT PARA 'CODICE' ;definisce il segmento di codice
PROCEDURA PROC      FAR             ;inizio della procedura
           ASSUME   CS:CODICE,DS:DATI,SS:STACK
           PUSH     DS              ;salva DS sullo stack
           SUB      AX,AX           ;azzera AX
           PUSH     AX              ;salva 0 sullo stack
           MOV      AX,DATI         ;indirizzo di DATI in AX
           MOV      DS,AX           ;indirizzo di DATI in DS
           CREA                     ;crea un file
           RET                      ;il controllo ritorna al DOS
PROCEDURA ENDP                      ;fine della procedura
CODICE     ENDS                     ;fine del segmento di codice
           END                      ;fine del programma
```

Figura 8.6 Creazione di un file tramite i servizi DOS

me realizzare questa operazione utilizzando i comandi DOS. Il corpo principale del programma, riportato qui di seguito, invoca semplicemente la macro CREA e il problema è risolto.

```
CREA   MACRO                    ;;macro per creare un file
       MOV      AH,3CH          ;;parametro di INT 21H per creare un file
       MOV      CX,00H          ;;attributo di file normale
       LEA      DX,NOMEFILE     ;;disco/nome/estensione del file da creare
       INT      21H             ;;crea il file
       ENDM
```

CREA (creazione di un file) inizializza i parametri perché venga eseguita l'apertura o la creazione di un file da parte dell'interruzione DOS 21H. Il registro CX contiene l'attributo del file.

0H	File normale
1H	File a sola lettura
2H	File non visibile
4H	File di sistema
8H	Etichetta del disco
10H	Sottodirettorio
20H	File archivio

La variabile NOMEFILE referenzia, nel segmento dati, una stringa ASCII che specifica l'indicazione del drive, il nome del file e l'estensione del file.

```
DATI        SEGMENT   PARA 'DATI'
NOMEFILE    DB        'B:NUMERI.DAT',0    ;nome del file da creare
DATI        ENDS
```

Il prossimo programma (Figura 8.7) permette all'utente di inserire da tastiera nomi e numeri di telefono in un file chiamato NUMERI.DAT, che è memorizzato nel dischetto contenuto nel drive B. Il delimitatore '0', che segue l'estensione del file, marca la fine della stringa. La macro CREA, con AH=3CH, esegue l'apertura o la creazione di un file e inizializza a 0 la lunghezza del file, per cui può essere invocata solo per creare un file nuovo oppure per cancellare un file esistente. Il descrittore del file, se necessario, viene restituito nel registro AX. Utilizzando le conoscenze acquisite con i programmi precedenti, potete senza grandi difficoltà scrivere un programma che permetta all'utente di modificare direttamente da tastiera i dati in un file.

Problema 7: Apertura, scrittura e chiusura di un file su disco

Il prossimo programma utilizza quattro macro, simili a quella definita nella Figura 8.6, per aprire, posizionarsi sulla fine del file, scrivere e chiudere un file. In Figura 8.7 viene riportato il codice sorgente del programma che permette di accedere al file precedentemente creato.
Per leggere da – o scrivere in – un file, è indispensabile averlo prima aperto. L'apertura di un file può essere eseguita mediante chiamata di interruzione, inizializzando il registro AH a 3CH. In questo caso, la lunghezza del file è zero, ad indicare che si tratta di un file vuoto. Per creare un file non occorre eseguire altre operazioni, mentre per aprire un file precedentemente creato è sufficiente aggiornare il contenuto del registro AH al valore 3DH.

```
APERTURA    MACRO                       ;;apre un file precedentemente creato
            MOV     AL,01H              ;;file a sola scrittura
            MOV     AH,3DH              ;;parametro per l'apertura
            LEA     DX,NOMEFILE         ;;disco/nome/est del file da aprire
            INT     21H                 ;;apertura del file
            MOV     DESFILE,AX          ;;salva il descrittore del file restituito
            ENDM
```

```
;per macchine IBM 8088/80386
;programma che esegue l'apertura, la scrittura e la chiusura di
;un file precedentemente creato

PAGE ,132                           ;dimensione pagina 66 per 132

IF1                                 ;include una libreria di macro
          INCLUDE C:MACLIB.MAC
ENDIF

APERTURA  MACRO                     ;;apre un file precedentemente creato
          MOV     AL,01H            ;;file a sola scrittura
          MOV     AH,3DH            ;;parametro per l'apertura
          LEA     DX,NOMEFILE       ;;disco/nome/est del file da aprire
          INT     21H               ;;apertura del file
          MOV     DESFILE,AX        ;;salva il descrittore del file restituito
          ENDM

FINEFILE  MACRO                     ;;si posiziona sulla fine del file
          MOV     AL,02H            ;;parametri per la chiamata DOS 21H
          MOV     AH,42H
          MOV     BX,DESFILE
          MOV     CX,0H
          MOV     DX,0H
          INT     21H               ;;calcolo dell'indice
          ENDM

SCRITTURA MACRO                     ;;permette di scrivere in un file
          MOV     AH,40H            ;;parametro per la scrittura
          MOV     BX,DESFILE        ;;descrittore file in BX
          MOV     CX,80             ;;numero di byte da scrivere
          LEA     DX,NOMEFILE       ;;punta ai caratteri da scrivere
          INT     21H               ;;scrittura
          ENDM

CHIUSURA  MACRO                     ;;chiude un file
          MOV     AH,3EH            ;;parametro per la chiusura
          MOV     BX,DESFILE        ;;descrittore file in BX
          INT     21H               ;;chiusura
          ENDM

STACK     SEGMENT PARA STACK   ;segmento di stack del programma
          DB      48 DUP ('STACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
STRINGA   LABEL   BYTE
LUNGMAX   DB      81            ;lunghezza massima dell'informazione + 1
LUNGREALE DB      ?             ;lunghezza reale dell'informazione
INFO      DB      80 DUP (' '),'$' ;informazione da memorizzare su disco
NOMEFILE  DB      'B:NUMERI.DAT',0 ;nome del file su cui scrivere
DESFILE   DW      ?             ;locazione per il descrittore file
PROMPT    DB      'NOME                                    TELEFONO$'
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE' ;definisce il segmento di codice
PROCEDURA PROC    FAR           ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS            ;salva DS sullo stack
          SUB     AX,AX         ;azzera AX
```

```
          PUSH     AX            ;salva 0 sullo stack
          MOV      AX,DATI       ;indirizzo di DATI in AX
          MOV      DS,AX         ;indirizzo di DATI in DS
          MOV      ES,AX         ;indirizzo di DATI in ES

          CANCELLA               ;cancella lo schermo
          APERTURA               ;apre il file
          FINEFILE
ANCORA:
          CALL     PASSAINFO     ;trasferisce su disco l'informazione da tastiera
          CMP      LUNGMAX,00    ;se manca l'informazione, fine del programma
          JNE      ANCORA        ;altrimenti, passa all'informazione successiva
          CHIUSURA               ;chiude il file
          RET
PROCEDURA ENDP

PASSAINFO PROC     NEAR          ;trasferisce l'informazione al file su disco
          CURSORE 0000H          ;posiziona il cursore in cima allo schermo
          STAMPACHAR PROMPT      ;richiesta di informazione all'utente
          CURSORE 0100H          ;aggiorna la posizione del cursore, per la scrittura
          MOV      AH,0AH        ;parametro per lettura stringa in ingresso
          LEA      DX,STRINGA    ;stringa in 'STRINGA'
          INT      21H           ;lettura da tastiera
          CANCELLA               ;cancella l'ultimo nome
          CMP      LUNGREALE,00  ;fine dell'informazione in ingresso?
          JE       FINE          ;se si, terminazione del programma
          MOV      BH,00         ;lunghezza della stringa in BX
          MOV      BL,LUNGREALE
          MOV      INFO[BX],' '  ;preparativi per trasferimento stringa su disco
          SCRITTURA              ;scrittura su disco

          CLD                    ;inizializza il flag direzione
          LEA      DI,INFO       ;preparativi per memorizzare i caratteri spazio
          MOV      CX,80         ;nella variabile INFO
          MOV      AL,20H        ;valore del carattere spazio (20H)
          REP      STOSB         ;ripetizione della memorizzazione per 80 volte

FINE:
          RET
PASSAINFO ENDP                   ;fine della procedura PASSAINFO

CODICE    ENDS                   ;fine del segmento di codice
          END                    ;fine del programma
```

Figura 8.7 Programma che permette di aprire, scrivere e chiudere un file precedentemente creato

Nella macro APERTURA, il contenuto di AL stabilisce se il file è a sola lettura (AL=0), a sola scrittura (AL=1) oppure se è lecito eseguire su di esso sia l'operazione di lettura che quella di scrittura (AL=2). Questa informazione occupa i tre bit meno significativi del registro AL, mentre i cinque bit più significativi, a partire dalla versione DOS 3.0, memorizzano informazioni per la condivisione dei file in rete locale. NOMEFILE deve identificare il file creato precedentemente, per cui deve contenere lo stesso indicatore di drive, lo stesso nome di file, la stessa estensione e lo stesso delimitatore (di un byte)

scelti in fase di creazione del file. NOMEFILE è una stringa di caratteri ASCII allocata nel segmento dati. Il descrittore di file, memorizzato nel registro AX dopo la chiamata dell'interruzione INT 21H, viene salvato nella variabile DESFILE, anch'essa definita nel segmento dati, per essere disponibile in futuro.
Prima di scrivere sul file, è necessario posizionarsi alla fine del file (EOF: End Of File), altrimenti il programma può sovrapporre l'informazione da memorizzare all'informazione precedentemente salvata sul file. A questo compito viene predisposta la macro FINEFILE:

```
FINEFILE   MACRO                  ;;si posiziona sulla fine del file
           MOV      AL,02H        ;;parametri per la chiamata DOS 21H
           MOV      AH,42H
           MOV      BX,DESFILE
           MOV      CX,0H
           MOV      DX,0H
           INT      21H           ;;calcolo dell'indice
           ENDM
```

Quando il registro AH contiene il valore 42H, il DOS permette di aggiornare la posizione del puntatore al file. Prima di invocare l'interruzione INT 21H, è indispensabile caricare il registro BX con il descrittore del file precedentemente salvato nella variabile DESFILE e il registro AL con la locazione iniziale referenziata dal puntatore al file. Se AL=0, l'offset viene calcolato dall'inizio del file; se AL=1, l'offset viene calcolato dalla posizione corrente, mentre se AL=2, l'offset viene determinato a partire dalla fine del file. Il valore di quest'offset è contenuto nella coppia di registri CX:DX. CX di solito memorizza il valore 0, a meno che il file non abbia una dimensione superiore a 64 kB. Quando AL=2, il programma cerca la fine del file, per cui i registri CX e DX contengono il valore 0. Dopo la chiamata dell'interruzione INT 21H, la coppia di registri DX:AX contiene il valore dell'offset calcolato (nel nostro esempio, il puntatore al file è stato posizionato alla fine del file).
Quando intendete scrivere su un file già aperto, utilizzando le funzioni DOS, la macro SCRITTURA vi permette di inizializzare i parametri DOS necessari per effettuare l'operazione voluta.

```
SCRITTURA  MACRO                  ;;permette di scrivere in un file
           MOV      AH,40H        ;;parametro per la scrittura
           MOV      BX,DESFILE    ;;descrittore file in BX
           MOV      CX,80         ;;numero di byte da scrivere
           LEA      DX,NOMEFILE   ;;punta ai caratteri da scrivere
           INT      21H           ;;scrittura
           ENDM
```

Infatti, AH=40H costituisce il parametro DOS che permette di scrivere su un file. Il registro BX contiene ancora il descrittore del file precedentemen-

te salvato nella variabile DESFILE, mentre CX memorizza il numero totale di byte da scrivere (il programma, ogni volta che viene eseguito, scrive su file un'intera linea di testo). L'indirizzo della stringa di byte che deve essere scritta è referenziato dalla coppia di registri DS:DX. In questo caso, INFO è memorizzata nel segmento dati e inizialmente contiene 80 caratteri spazio e il carattere di tappo $, come qui di seguito indicato:

```
DATI         SEGMENT   PARA 'DATI'
STRINGA      LABEL     BYTE
LUNGMAX      DB        81                  ;lunghezza massima
                                           ;dell'informazione + 1
LUNGREALE    DB        ?                   ;lunghezza reale
                                           ;dell'informazione
INFO         DB        80 DUP (' '),'$'    ;informazione
                                           ;da memorizzare su disco
NOMEFILE     DB        'B:NUMERI.DAT',0    ;nome del file su cui scrivere
DESFILE      DW        ?                   ;locazione per il descrittore
                                           ;file
PROMPT       DB        'NOME                          TELEFONO$'
DATI         ENDS
```

Quando il programma termina di scrivere i dati nel file, tutti i file che erano stati aperti vengono chiusi, invocando la macro CHIUSURA con AH=3EH.

```
CHIUSURA     MACRO                         ;;chiude un file
             MOV       AH,3EH              ;;parametro per la chiusura
             MOV       BX,DESTFILE         ;;descrittore file in BX
             INT       21H                 ;;chiusura
             ENDM
```

Si noti come sia ancora necessario specificare il descrittore del file, ma non l'informazione NOMEFILE. In altre parole, BX deve referenziare il descrittore dell'ultimo file aperto.

Da una attenta analisi, ci si rende conto che questo programma utilizza avanzati comandi di stringa (ad esempio REP STOSB), ottimizzando così il numero delle istruzioni di codice:

```
             CANCELLA                      ;cancella lo schermo
             APERTURA                      ;apre il file
             FINEFILE
ANCORA:
             CALL      PASSAINFO           ;trasferisce su disco
                                           ;l'informazione da tastiera
             CMP       LUNGMAX,00          ;se manca l'informazione,
                                           ;fine del programma
             JNE       ANCORA              ;altrimenti, passa
                                           ;all'informazione successiva
             CHIUSURA                      ;chiude il file
             RET
PROCEDURA    ENDP
```

Lo schermo viene cancellato dalla macro CANCELLA che è contenuta nella libreria MACLIB.MAC. Viene aperto il file B:NUMERI.DAT (questo file era stato creato dal programma di Figura 8.6) e il puntatore al file viene posizionato in fondo al file dalla macro FINEFILE. Il file, in questo modo, è pronto a memorizzare altre informazioni oltre a quelle che già contiene. La procedura PASSAINFO permette all'utente di inserire da tastiera i caratteri da memorizzare nel file, che vengono salvati nella variabile INFO. Se l'informazione inserita da tastiera non contiene altro che il carattere di ritorno a capo, il programma termina l'esecuzione dopo aver invocato la macro CHIUSURA.
Analizzando dettagliatamente la procedura PASSAINFO, è possibile fare alcune osservazioni:

```
PASSAINFO   PROC         NEAR      ;trasferisce l'informazione al file
                                   ;su disco
            CURSORE      0000H     ;posiziona il cursore in cima
                                   ;allo schermo
            STAMPACHAR   PROMPT    ;richiesta di informazione all'utente
            CURSORE      0100H     ;aggiorna la posizione del cursore,
                                   ;per la scrittura
```

In queste poche linee di codice, il cursore viene posizionato dalla macro CURSORE, contenuta nella libreria MACLIB.MAC, in alto a sinistra sullo schermo. La macro STAMPACHAR, contenuta nella stessa libreria, visualizza in cima allo schermo un messaggio all'utente, a cui richiede un nome e un numero di telefono. L'utente deve inserire da tastiera questa informazione (fino a 80 byte), dopo che il cursore è stato spostato all'inizio della linea successiva.

```
            MOV          AH,0AH           ;parametro per lettura stringa
                                          ;in ingresso
            LEA          DX,STRINGA       ;stringa in 'STRINGA'
            INT          21H              ;lettura da tastiera
            CANCELLA                      ;cancella l'ultimo nome
            CMP          LUNGREALE,00     ;fine dell'informazione in
                                          ;ingresso?
            JE           FINE             ;se sì, terminazione del
                                          ;programma
            MOV          BH,00            ;lunghezza della stringa in BX
            MOV          BL,LUNGREALE
            MOV          INFO[BX],' '     ;preparativi per trasferimento
                                          ;stringa su disco
            SCRITTURA                     ;scrittura su disco
```

La dichiarazione STRINGA LABEL BYTE, presente nel segmento dati, permette di salvare nella variabile INFO fino a 80 caratteri inseriti da tastiera. Si ricordi che l'interruzione INT 21H restituisce la lunghezza della stringa

nel primo byte, per cuï in LUNGMAX viene memorizzato il valore 81 (byte). Una volta che la stringa viene inserita da tastiera e salvata nella variabile INFO, lo schermo viene cancellato per permettere ulteriori ingressi da tastiera. Viene effettuato un controllo sul valore di LUNGREALE, per verificare se l'ingresso da tastiera contiene effettivamente un'informazione oppure il solo carattere di ritorno a capo. La lunghezza reale di questa informazione viene trasferita nel registro BX e il carattere di ritorno a capo viene sostituito da un carattere spazio (' '). La macro SCRITTURA trasferisce il testo dell'informazione su disco e il puntatore al file automaticamente si sposta alla fine del file.
Prima di richiedere un altro ingresso da tastiera, vengono eseguite alcune operazioni di cancellazione:

```
CLD                 ;inizializza il flag direzione
LEA     DI,INFO     ;preparativi per memorizzare i caratteri
                    ;spazio
MOV     CX,80       ;nella variabile INFO
MOV     AL,20H      ;valore del carattere spazio (20H)
REP     STOSB       ;ripetizione della memorizzazione per 80
                    ;volte
```

INFO contiene ancora l'informazione precedente. Se questa variabile non venisse reinizializzata con 80 caratteri spazio (20H), mediante l'istruzione REP STOSB, e se la successiva stringa in ingresso fosse di dimensione minore della precedente, si verificherebbe una sovrapposizione indesiderata di due informazioni distinte.
Il ciclo codificato nella procedura PASSAINFO viene ripetuto fino a quando l'utente preme solo il tasto di ritorno a capo. Il programma, allora, termina e il file viene chiuso. Il contenuto di questo file può essere listato, invocando il seguente comando DOS:

```
A>TYPE B:NUMERI.DAT
```

Il prossimo programma illustra un metodo per leggere il contenuto di un file, a livello di linguaggio assemblatore, invocando l'interruzione DOS INT 21H.

Problema 8: Apertura, lettura e chiusura di un file su disco

Negli ultimi due programmi, abbiamo creato un file di dati e abbiamo permesso all'utente di scrivere in esso. In questo programma, il file precedentemente creato viene aperto e ne viene letto il contenuto. La Figura 8.8 mostra il programma che garantisce l'accesso a questo file.
Il programma utilizza tre macro e una procedura di tipo NEAR. Riportiamo

```
;per macchine IBM 8088/80386
;programma che esegue la lettura di un file precedentemente
;salvato su disco

PAGE ,132                         ;dimensione pagina 66 per 132

IF1                               ;include una libreria di macro
        INCLUDE C:MACLIB.MAC
ENDIF

APERTURA  MACRO                   ;;apre un file precedentemente creato
          MOV     AL,00H          ;;file a sola scrittura
          MOV     AH,3DH          ;;parametro per l'apertura
          LEA     DX,NOMEFILE     ;;disco/nome/est del file da aprire
          INT     21H             ;;apertura del file
          MOV     DESFILE,AX      ;;salva il descrittore del file restituito
          ENDM

LETTURA   MACRO                   ;;legge un record del file
          MOV     AH,3FH          ;;parametro per la lettura
          MOV     BX,DESFILE      ;;descrittore del file in BX
          MOV     CX,80           ;;numero di byte da leggere
          LEA     DX,INFO         ;;caratteri da leggere
          INT     21H             ;;lettura
          ENDM

CHIUSURA  MACRO                   ;;chiude un file
          MOV     AH,3EH          ;;parametro per la chiusura
          MOV     BX,DESFILE      ;;descrittore file in BX
          INT     21H             ;;chiusura
          ENDM

STACK     SEGMENT PARA STACK   ;segmento di stack del programma
          DB      48 DUP ('STACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
STRINGA   LABEL   BYTE
LUNGMAX   DB      81              ;lunghezza massima dell'informazione + 1
LUNGREALE DB      ?               ;lunghezza reale dell'informazione
INFO      DB      80 DUP (' '),'$' ;informazione da memorizzare su disco
NOMEFILE  DB      'B:NUMERI.DAT',0 ;nome del file su cui scrivere
DESFILE   DW      ?               ;locazione per il descrittore file
PROMPT    DB      'NOME                                   TELEFONO$'
POSIZIONE DW      0100H
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE' ;definisce il segmento di codice
PROCEDURA PROC    FAR             ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS              ;salva DS sullo stack
          SUB     AX,AX           ;azzera AX
          PUSH    AX              ;salva 0 sullo stack
          MOV     AX,DATI         ;indirizzo di DATI in AX
          MOV     DS,AX           ;indirizzo di DATI in DS

          CANCELLA                ;cancella lo schermo
          APERTURA                ;apre il file

ANCORA:
          CALL    PASSAINFO       ;trasferisce su disco l'informazione da tastiera
```

```
         CMP      LUNGMAX,00     ;se manca l'informazione, fine del programma
         JNE      ANCORA         ;altrimenti, passa all'informazione successiva
         CHIUSURA                ;chiude il file
         RET
PROCEDURA ENDP

PASSAINFO PROC    NEAR           ;trasferisce l'informazione al file su disco
         CURSORE 0000H           ;posiziona il cursore in cima allo schermo
         STAMPACHAR PROMPT       ;richiesta di informazione all'utente
CICLO:   CURSORE POSIZIONE       ;aggiorna la posizione del cursore, per la scrittura
         LETTURA                 ;legge il file
         MOV      LUNGREALE,AL
         CMP      LUNGREALE,00   ;fine dell'informazione in ingresso?
         JE       FINE           ;se si, terminazione del programma
         STAMPACHAR STRINGA+2
         ADD      POSIZIONE,0100H
         JMP      CICLO
FINE:
         RET
PASSAINFO ENDP                   ;fine della procedura PASSAINFO

CODICE   ENDS                    ;fine del segmento di codice
         END                     ;fine del programma
```

Figura 8.8 Lettura di un file salvato precedentemente

qui di seguito le macro APERTURA (apre il file), LETTURA (legge il file) e CHIUSURA (chiude il file):

```
APERTURA  MACRO                  ;;apre un file precedentemente creato
          MOV    AL,00H          ;;file a sola lettura
          MOV    AH,3DH          ;;parametro per l'apertura
          LEA    DX,NOMEFILE     ;;disco/nome/est del file da aprire
          INT    21H             ;;apertura del file
          MOV    DESFILE,AX      ;;salva il descrittore del file restituito
          ENDM
LETTURA   MACRO                  ;;legge un record del file
          MOV    AH,3FH          ;;parametro per la lettura
          MOV    BX,DESFILE      ;;descrittore del file in BX
          MOV    CX,80           ;;numero di byte da leggere
          LEA    DX,INFO         ;;caratteri da leggere
          INT    21H             ;;lettura
          ENDM
CHIUSURA  MACRO                  ;;chiude un file
          MOV    AH,3EH          ;;parametro per la chiusura
          MOV    BX,DESFILE      ;;descrittore file in BX
          INT    21H             ;;chiusura
          ENDM
```

Analizzando dettagliatamente le macro APERTURA e CHIUSURA, si può notare che si tratta delle stesse macro utilizzate nel precedente programma, per aprire, scrivere e chiudere un file. La macro LETTURA, con AH=3FH,

permette all'utente di leggere il contenuto del file, in base all'informazione memorizzata nel descrittore di file e trasferita nel registro BX. Il registro CX, come al solito, specifica il numero di byte da leggere e l'indirizzo di INFO viene trasferito nel registro DX. Il registro DS referenzia il segmento dati. INFO è stata definita in memoria con una dimensione sufficiente a soddisfare le richieste di lettura indicate dal contenuto del registro CX. Dopo che è stata invocata l'interruzione DOS 21H, il registro AX contiene il numero di byte letti; se AX è a 0, in fase di lettura è stato incontrato il marcatore di fine file.
Il segmento dati definito in questo programma è simile a quello di Figura 8.7. Questa non è una sorpresa, in quanto l'informazione che l'utente intende leggere dal file presenta lo stesso formato dell'informazione che egli aveva precedentemente inserito da tastiera.

```
DATI        SEGMENT   PARA 'DATI'
STRINGA     LABEL     BYTE
LUNGMAX     DB        81                 ;lunghezza massima
                                         ;dell'informazione + 1
LUNGREALE   DB        ?                  ;lunghezza reale
                                         ;dell'informazione
INFO        DB        80 DUP (' '),'$'   ;informazione
                                         ;da memorizzare su disco
NOMEFILE    DB        'B:NUMERI.DAT',0   ;nome del file su cui scrivere
DESFILE     DW        ?                  ;locazione per il descrittore
                                         ;file
PROMPT      DB        'NOME                          TELEFONO$'
POSIZIONE   DW        0100H
DATI        ENDS
```

Si ricordi che l'ultimo byte contenuto in NOMEFILE memorizza il carattere tappo che indica la fine del nome del file da aprire. Il contenuto della variabile PROMPT è identico a quello definito nel programma precedente.
Il codice che costituisce il corpo della procedura principale del programma è chiaro e conciso:

```
          CANCELLA                    ;cancella lo schermo
          APERTURA                    ;apre il file
ANCORA:
          CALL      PASSAINFO         ;trasferisce su disco l'informazione
                                      ;da tastiera
          CMP       LUNGMAX,00        ;se manca l'informazione, fine del
                                      ;programma
          JNE       ANCORA            ;altrimenti, passa all'informazione
                                      ;successiva
          CHIUSURA                    ;chiude il file
```

CANCELLA è una macro che appartiene alla libreria MACLIB.MAC ed ese-

gue la cancellazione dello schermo, prima che il file venga aperto dalla macro APERTURA.
Se l'operazione di apertura del file viene portata correttamente a termine, la chiamata alla procedura PASSAINFO restituisce l'informazione del file richiesta dall'utente. Se questa procedura non restituisce nulla, LUNGREAL assume il valore 0 e il programma termina dopo l'invocazione della macro CHIUSURA. Riportiamo qui di seguito il codice della procedura PASSAINFO:

```
PASSAINFO  PROC          NEAR             ;trasferisce l'informazione
                                          ;al file su disco
           CURSORE       0000H            ;posiziona il cursore in cima
                                          ;allo schermo
           STAMPACHAR    PROMPT           ;richiesta di informazione
                                          ;all'utente
CICLO:     CURSORE       POSIZIONE        ;aggiorna la posizione del
                                          ;cursore, per la scrittura
           LETTURA                        ;legge il file
           MOV           LUNGREALE,AL
           CMP           LUNGREALE,00     ;fine dell'informazione in
                                          ;ingresso?
           JE            FINE             ;se sì, terminazione del
                                          ;programma
           STAMPACHAR    STRINGA+2
           ADD           POSIZIONE,0100H
           JMP           CICLO
FINE:
```

La procedura PASSAINFO invoca diverse macro che sono contenute nella libreria MACLIB.MAC. La macro CURSORE sposta il cursore nell'angolo in alto a sinistra dello schermo; la macro STAMPACHAR visualizza sullo schermo – in corrispondenza della posizione corrente del cursore – il contenuto della variabile PROMPT che è stata definita nel segmento dati del programma. La posizione del cursore viene poi aggiornata in base al valore della variabile POSIZIONE, valore che cambia ogni volta che il ciclo viene eseguito, in modo tale da visualizzare le informazioni su linee distinte dello schermo.
La macro LETTURA, come abbiamo già detto, legge dal file l'informazione richiesta e la memorizza nella struttura dati STRINGA. Viene eseguito un controllo di terminazione del file per stabilire se continuare o meno l'esecuzione del ciclo. La macro STAMPACHAR, contenuta nella libreria MACLIB.MAC, visualizza sullo schermo l'informazione contenuta nella struttura dati STRINGA, a partire dal terzo byte in memoria. Si tenga presente che il primo byte contiene l'indicazione della lunghezza massima dell'informazione, mentre il secondo contiene la lunghezza reale. Il valore della variabile POSIZIONE viene poi incrementato per posizionare il cursore sulla linea successiva e il programma continua.
Se la lista dei numeri di telefono è lunga, questa viene visualizzata fino a

utilizzare l'ultima linea dello schermo, dopo di che non compare più nulla (è un utile esercizio ampliare il programma in modo da correggere questa limitazione).
Siete ora in grado di scrivere un programma a menu che esegua le funzioni descritte negli ultimi tre esempi. Il menu deve essere realizzato in modo tale che fornisca all'utente le seguenti opzioni:

1. Creazione di un file
2. Scrittura in un file
3. Lettura da un file
4. Abbandono del programma

Se l'utente seleziona le opzioni 1, 2 oppure 3, il programma richiede l'indicazione del drive, il nome del file e l'estensione del nome del file. In questo modo, è possibile creare, scrivere e leggere file senza ricorrere a programmi diversi.

8.7 Programmazione in modalità reale e virtuale protetta

La programmazione in modalità di indirizzamento reale e virtuale è stata introdotta con il microprocessore Intel 80286. I precedenti microprocessori, come ad esempio l'8086, erano in grado di operare solo in modalità di indirizzamento reale, per cui il programmatore poteva trattare solo indirizzi fisici. Quando l'80286 opera in modalità di indirizzamento reale, può accedere a 16 777 216 (0FFFFFFH + 1) locazioni di memoria, in quanto il suo bus indirizzi si compone di 24 linee, mentre quando opera in modalità di indirizzamento virtuale, può accedere a un gigabyte di memoria virtuale (logica).
La modalità di indirizzamento virtuale permette di trasferire blocchi di codice e di dati dalla memoria reale ad una memoria secondaria (come ad esempio un disco rigido) e viceversa, in modo trasparente al programmatore. Questa gestione della memoria è integrata direttamente sul chip per i microprocessori 80286 e 80386. L'80286 può accedere ad una grande quantità di memoria, ma l'80386, che dispone di un bus indirizzi di 32 bit, può accedere addirittura a 4 294 967 296 (0FFFFFFFFH + 1) – cioè quattro gigabyte – locazioni di memoria fisica. Inoltre, l'80386 è in grado di referenziare 64 terabyte di memoria virtuale (una enormità di spazio in confronto alle prime macchine che disponevano di una memoria di 64 kB).
Per accedere a questa grande quantità di memoria, sono stati codificati sistemi operativi particolarmente efficienti su macchine con 640 kB di memoria centrale. Infatti, non è stato possibile accedere ad una memoria superiore

a 640 kB, se non tramite un drive virtuale, con la versione DOS 3.2. Questa limitazione penalizza i programmi – come Topview dell'IBM – che necessitano di una grande quantità di memoria (fisica e non virtuale! L'accesso alla memoria virtuale costituisce un'altra sfida ai sistemisti).
Non è un compito semplice passare dalla modalità di indirizzamento reale a quella virtuale e viceversa. Anche a livello di linguaggio assemblatore, questa operazione non è supportata direttamente. Per avere maggiori dettagli sull'accesso alla memoria virtuale, consultate il Capitolo 9 di *iAPX 286 Operating Systems Writer's Guide* (121960 – 001) e il Capitolo 4 di *iAPX 386 High-Performance Microprocessor with Integrated Memory Management* (231630 – 001), in quanto lo scopo di questo libro non è quello di esaminare dettagliatamente come operare in modalità di indirizzamento virtuale.
Nella pubblicazione di *PC Tech Journal* dell'Agosto 1985, Guy Quedens e Gary Webb scrissero un articolo intitolato *Switching Modes*, attraverso cui spiegavano come passare dalla modalità di indirizzamento reale alla modalità di indirizzamento virtuale. Il loro programma, scritto in linguaggio assemblatore 80286, permetteva anche di ritornare alla modalità di indirizzamento reale ed è stato da noi lievemente modificato per adattarlo al monitor grafico a colori (Figura 8.9).

```
;solo per macchine 80286/80386
;programma che indica come entrare in modalità virtuale e come
;abbandonarla

COMMENT *
Questo programma è stato estratto dalla pubblicazione "PC TECH
JOURNAL" dell'Agosto 1985 (Copyright 1985 by Ziff-Davis
Publishing Company). Il programma originale e stato scritto da
Guy Quedens e Gary Webb e realizza il passaggio in modalità di
indirizzamento virtuale, modifica gli attributi di
visualizzazione e ripristina la modalità di indirizzamento reale
prima di passare il controllo al DOS. E' stato modificato per
adattarlo al monitor grafico a colori.

---------> Produce un file .COM <-----------

ATTENZIONE: Questo programma può essere eseguito solo su un
calcolatore IBM AT. *


PAGE    ,132

seg_dati_bios SEGMENT at 0040h
              ORG       0067h
iniz_rom_io   dw ?         ;due variabili doubleword nel segmento dati BIOS
seg_rom_io    dw ?         ;usate per memorizzare un indirizzo di 32 bit
seg_dati_bios ENDS

descrittore   STRUC
limite_seg    dw 0         ;dimensione del segmento (da 1 a 65536 byte)
```

```
base_lsb        dw 0         ;indirizzo fisico di 24 bit
base_msb        dw 0         ;(da 0 a (16Mbyte - 1))
dir_acc         db 0         ;byte dei diritti di accesso
                dw 0         ;riservato all'80386
descrittore     ENDS

porta_cmos      equ 070h
accesso_cod     equ 10011011b ;byte dei diritti di accesso a segmenti di codice
accesso_dati    equ 10010011b ;byte dei diritti di accesso a segmenti di dati
disab_bit20     equ 11011101b ;codice funzione 8042 per disabilitare bit 20
abili_bit20     equ 11011111b ;codice funzione 8042 per abilitare bit 20
inta01          equ 021h      ;Controllore Interruzioni 8259 #1
intb01          equ 0A1h      ;Controllore Interruzioni 8259 #2
porta_a         equ 060h      ;porta A dell'8042
shut_cmd        equ 0FEh      ;comando all'8042: disattiva l'AT
shut_down       equ 00Fh      ;indice al byte di disattivazione CMOS
porta_stato     equ 064h      ;porta di stato dell'8042
virtuale        equ 0001h     ;LSB = 1: modalità virtuale protetta

;*****************************************************************************
;MACRO NECESSARIA SE NON SUPPORTATA DALL'ASSEMBLER (IL MACROASSEMBLER IBM 2.0
;NON LA SUPPORTA, AL CONTRARIO DEL TURBO ASSEMBLER SPEEDWARE!)

lgdt    MACRO   lgdt1       ;;carica la tabella di descrittori globale
        LOCAL   lgdt2,lgdt3
        db      00FH

lgdt2   label   byte
        mov     dx,word ptr lgdt1
lgdt3   label   byte
        org     offset lgdt2
        db      001h
        org     offset lgdt3
        ENDM

lmsw    MACRO   lmsw1       ;;carica la parola di stato della macchina
        LOCAL   lmsw2,lmsw3
        db      00FH

lmsw2   label   byte
        mov     si,ax
lmsw3   label   byte
        org     offset lmsw2
        db      001h
        org     offset lmsw3
        ENDM
;*****************************************************************************

jmpfar  MACRO   jmpfar1,jmpfar2
        db      0EAh
        dw      (offset jmpfar1)
        dw      jmpfar2
        ENDM

segcod  SEGMENT para public 'code'
        ASSUME  cs:segcod
        org     100h
inizio: jmp     short main
EVEN
gdt     LABEL   word
desc_gdt     EQU  (($-gdt)/8)*8 + 0000000000000000b
```

```
gdt1        descrittore <lung_gdt,,,accesso_dati,>
cod_cs      EQU  (($-gdt)/8)*8 + 0000000000000000b
gdt_2       descrittore <lung_segcod,,,accesso_cod,>
dati_cs     EQU  (($-gdt)/8)*8 + 0000000000000000b
gdt3        descrittore <lung_segcod,,,accesso_dati,>
desc_ss     EQU  (($-gdt)/8)*8 + 0000000000000000b
gdt4        descrittore <0FFFFh,,,accesso_dati,>
desc_ds     EQU  (($-gdt)/8)*8 + 0000000000000000b
gdt5        descrittore <0FFFFh,,,accesso_dati,>
desc_es     EQU  (($-gdt)/8)*8 + 0000000000000000b
gdt6        descrittore <0FFFFh,,,accesso_dati,>
lung_gdt    EQU  $-gdt
PAGE

;----------------------------------------------------------------------:
; Formato del campo Selettore di segmento (nell'indirizzo virtuale):   :
;                                                                      :
; +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+                    :
;                   INDICE                  TI  RPL                    :
; +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+                    :
;                                                                      :
; TI = Indicatore di tabella (0=GDT, 1=LDT)                            :
; RPL = Livello di privilegio richiesto (00=maggiore, 11=minore)       :
;----------------------------------------------------------------------:
; Formato della tabella di descrittori globale (GDT)                   :
;                                                                      :
;                     . -----------+                     +---> TI     :
;                     v                                   ++-> RPL     :
;      GDT ==> +---------------+                                       :
;                  DESC_GDT       --          0000000000000000b        :
;              +---------------+                                       :
;                   COD_CS                    0000000000001000b        :
;              +---------------+                                       :
;                  DATI_CS                    0000000000010000b        :
;              +---------------+                                       :
;                  DESC_SS                    0000000000011000b        :
;              +---------------+                                       :
;                  DESC_DS                    0000000000100000b        :
;              +---------------+                                       :
;                  DESC_ES                    0000000000101000b        :
;              +---------------+                                       :
;----------------------------------------------------------------------:

i8259_1 db ?            ;per memorizzare lo stato dell'8259 #1
i8259_2 db ?            ;per memorizzare lo stato dell'8259 #2

;----------------------------------------------------------------------:
; MAIN                                                                 :
;----------------------------------------------------------------------:

        ASSUME  ds:segcod
main    PROC                     ;ES=DS=CS
        cld                      ;in avanti
        mov     dx,cs            ;calcolo dell'indirizzo fisico
        mov     cx,offset gdt    ;di 24 bit da CS:GDT
        call    indirizzo_24bit
        mov     gdt1.base_lsb,dx ;DESC_GDT punta ora alla gdt
        mov     gdt1.base_msb,cl
        mov     dx,cs            ;calcolo dell'indirizzo fisico
        xor     cx,cx            ;di 24 bit da CS:0000
        call    indirizzo_24bit
        mov     gdt2.base_lsb,dx ;COD_CS punta ora a segcod come
```

```
        mov     gdt2.base_msb,cl ;ad un segmento di codice
        mov     gdt3.base_lsb,dx ;COD_CS punta ora a segcod come
        mov     gdt3.base_msb,cl ;ad un segmento di dati
        mov     dx,ss            ;calcolo dell'indirizzo fisico
        xor     cx,cx            ;di 24 bit da SS:0000
        call    indirizzo_24bit
        mov     gdt4.base_lsb,dx ;DESC_SS punta ora al
        mov     gdt4.base_msb,cl ;segmento di stack
        lgdt    gdt              ;carica il registro GDTR
        mov     ah,abili_bit20   ;abilitazione bit 20 bus indirizzi
        call    gate_a20
        or      al,al            ;è stato accettato il comando?
        jz      m_10             ;se si, continua
        mov     dx,offset errore ;stampa il messaggio di errore
        mov     ah,9             ;e termina
        int     21h
        int     20h
errore  db      "Address line A20 failed to Gate Opens$"
m_10    cli                      ;no interruzioni
        in      al,inta01        ;stato del Controllore di Interruzioni #1
        mov     i8259_1,al
        in      al,intb01        ;stato del Controllore di Interruzioni #2
        mov     i8259_1,al
        ASSUME  ds:seg_dati_bios
        mov     dx,seg_dati_bios ;indirizzo di ritorno modalità reale
        mov     ds,dx
        mov     seg_rom_io,cs
        mov     iniz_rom_io,offset reale
        mov     al,shut_down     ;preparativi per la sospensione
        out     porta_cmos,al
        jmp     short $+2        ;attesa I/O
        mov     al,5
        out     porta_cmos+1,al
        mov     ax,virtuale      ;definizione della parola di stato per
        lmsw    ax               ;passare in modalità virtuale
        jmpfar  m_20,cod_cs      ;deve disattivare la coda di prefetch
m_20    ASSUME  ds:segcod        ;IN MODALITA' VIRTUALE...
        mov     ax,desc_ss       ;selettore del segmento di stack
        mov     ss,ax            ;ss+sp dell'utente non è un descrittore
        mov     ax,dati_cs
        mov     ds,ax            ;DS = SEGCOD come dati
        mov     gdt5.base_lsb,8000h ;usa 0000 per MONITOR MONO
        mov     gdt5.base_msb,0Bh
        mov     gdt6.base_lsb,8000h ;usa 0000 per MONITOR MONO
        mov     gdt6.base_msb,0Bh
        mov     ax,desc_ds
        mov     ds,ax
        mov     ax,desc_es
        mov     es,ax
        mov     cx,80*25
        xor     si,si
        xor     di,di
m_30    lodsw
        mov     ah,70h           ;attributo per aggiornare il video
        stosw
        loop    m_30
        mov     al,shut_cmd      ;sospende
        out     porta_stato,al   ;ritorno in modalità REALE
m_40    hlt
        jmp     short m_40
```

```
;---------------------------------------------------------------------:
; GATE_A20                                                             :
;Questa routine controlla un segnale che attiva la linea 20 del bus:
;indirizzi. Il segnale A20 è una uscita del processore slave 8042: deve :
;essere attivato  prima di entrare in modalità protetta e disattivato :
;prima di uscire  dalla modalità protetta per entrare in modalità reale.   :
;Ingressi: (AH)=0DDh bit 20 bus indirizzi disattivato (A20 sempre 0)      :
;          (AH)=0DFh bit 20 bus indirizzi attivato    (286 controlla A20)  :
;Uscite:  (AL)=0 esito dell'operazione positivo. L'8042 ha accettato il :
;comando                                                   :
;(AL)=2 errore -- L'8042 non accetta il comando.           :
;---------------------------------------------------------------------:

gate_a20  PROC
        cli                         ;interruzioni disabilitate quando opera l'8042
        call    vuoto_8042          ;assicura buffer input 8042 vuoto
        jnz     gate_a20_01         ;ritorno se l'8042 non accetta il comando
        mov     al,0D1h             ;comando all'8042 per scrivere nella porta di uscita
        out     porta_stato,al      ;uscita comando all'8042
        call    vuoto_8042          ;attesa che l'8042 accetti il comando
        jnz     gate_a20_01         ;ritorno se l'8042 non accetta il comando
        mov     al,ah               ;dati alla porta dell'8042
        out     porta_a,al          ;uscita dati alla porta per l'8042
        call    vuoto_8042          ;attesa 8042 per dati alla porta
gate_a20_01:
        ret
gate_a20  ENDP

;---------------------------------------------------------------------:
; VUOTO_8042                                                           :
; Questa routine attende che il buffer dell'8042 si svuoti             :
; Ingressi: Nessuno                                                    :
; Uscite:   (AL)=0 buffer di ingresso 8042 vuoto (ZF=1)                :
;           (AL)=2 Attesa, buffer 8042 pieno (ZF=0)                    :
;---------------------------------------------------------------------:

vuoto_8042  PROC
        push    cx                  ;salva CX
        sub     cx,cx               ;CX=0 sarà il valore di attesa
vuoto_8042_01:
        in      al,porta_stato      ;lettura porta di stato dell'8042
        and     al,00000010b        ;test flag di riempimento buffer ingresso (D1)
        loopnz  vuoto_8042_01       ;ciclo fino a che il buffer di ingresso è vuoto
        pop     cx                  ;ripristino di CX
        ret
vuoto_8042  ENDP

;---------------------------------------------------------------------:
; INDIRIZZO_24BIT                                                      :
; Ingressi: DX referenzia un segmento                                  :
;           CX referenzia un offset                                    :
; Uscite:   DX contiene i bit meno significativi dell'indirizzo di base  :
;           CL contiene i bit più significativi dell'indirizzo di base   :
;---------------------------------------------------------------------:

indirizzo_24bit  PROC
        push    ax
        rol     dx,4
        mov     ax,dx
        and     dl,0F0h
        and     ax,0Fh
```

```
        add     dx,cx           ;calcolo dell'indirizzo a 24 bit
        mov     cx,ax           ;in CL byte meno significativo della base
        adc     cl,ch           ;carry (CH=0)
        pop     ax
        ret
indirizzo_24bit  ENDP

        ASSUME  ds:segcod       ;IN  MODALITA' REALE...
reale:  mov     dx,cs
        mov     ds,dx           ;DS=CS
        mov     ah,disab_bit20  ;disabilitazione bit 20 bus indirizzi
        call    gate_a20
        mov     al,i8259_1
        out     inta01,al       ;aggiorna stato del Controllore di Interruzioni #1
        mov     al,i8259_2
        out     intb01,al       ;aggiorna stato del Controllore di Interruzioni #2
        sti                     ;interruzioni abilitate
        int     20h             ;ritorno al DOS
main    ENDP
lung_segcod     EQU $
segcod  ENDS
        END     inizio
```

Figura 8.9 Programma che illustra come entrare in modalità protetta e uscirne (leggermente modificato rispetto al programma originale scritto da Guy Quedens e Gary Webb, *Switching Modes*, PC Tech, Agosto 1985)

9

Programmare con il coprocessore 80287/80387

La famiglia di componenti programmabili Intel 8087/80387 opera in parallelo con i microprocessori 8088/80386. Mentre l'8088/80386 è un microprocessore per applicazioni di tipo generale, l'8087/80387 si dedica ad elaborazioni numeriche e, poiché viene sempre utilizzato insieme al microprocessore 8088/80386, prende il nome di coprocessore (in particolare, l'80286 supporta l'attività del coprocessore 80287, mentre l'80386 supporta entrambi i coprocessori 80287 o 80387).
Tutta la programmazione in linguaggio assemblatore che abbiamo presentato finora ha trattato unicamente operazioni aritmetiche tra numeri interi. L'80287/80387 può eseguire queste operazioni anche sui numeri reali, senza ricorrere a particolari routine di conversione software. Le routine di conversione software sono disponibili da molto tempo e sono utilizzate da tutti i linguaggi di alto livello, come BASIC, APL, Fortran e Pascal. La funzione di queste routine è di convertire un numero reale (come 3.14159, 100.3456 oppure −4.565E12) in una sequenza di cifre che viene interpretata dall'80286/80386 come un numero intero. In questa forma, il numero viene utilizzato nelle operazioni aritmetiche e viene convertito nuovamente in un numero reale da un'apposita routine. Naturalmente queste operazioni di conversione rallentano l'esecuzione del programma, a tal punto che sono state codificate due versioni di BASIC: una in grado di eseguire solo operazioni su numeri interi (più veloce e usata spesso per i videogiochi) e un'altra in grado di eseguire operazioni sia sui numeri interi che sui numeri reali.
La Intel, però, riuscì a progettare un chip che eseguiva a livello hardware le operazioni sui numeri reali, con indubbi vantaggi in termini di velocità e precisione. Circa nello stesso periodo, la IEEE (Institute of Electrical and Electronic Engineers) definì una rappresentazione standard dei numeri rea-

li, che venne adottata dalla Intel e diventò da quel momento comune a tutti i sistemi di piccole dimensioni.
Oggi vengono utilizzati principalmente due formati per i numeri reali: il formato Microsoft, adottato dalla maggior parte dei linguaggi di alto livello che non supportano i coprocessori, e il formato IEEE (o Intel), adottato dal coprocessore 80287/80387. Installando l'80287/80387 su un sistema di piccole dimensioni, è possibile – una volta specificato il formato IEEE per i numeri reali – realizzare elaborazioni numeriche a livello hardware, anziché a livello software.
Nel linguaggio assemblatore, la direttiva .8087 permette di memorizzare i numeri reali nel formato IEEE, anziché nel formato Microsoft che non è interpretabile dai coprocessori. La conversione numerica nel formato reale viene eseguita in fase di traduzione, ma non esiste un processo analogo che permetta di convertire, in un formato comprensibile all'utente, i risultati reali elaborati dal coprocessore. È quindi necessario effettuare via software la conversione finale.

9.1 Specifiche del coprocessore

Il coprocessore 8087 è stato introdotto nel 1979 come estensione del microprocessore 8088/8086. Nel 1982, la Intel ha costruito il coprocessore 80287 e, tre anni dopo, ha realizzato il coprocessore 80387. Sia l'80287 che l'80387 dispongono di una architettura interna a 80 bit, adottano il formato in virgola mobile IEEE, supportano sette tipi di dati (reale in singola precisione a 32 bit, reale in doppia precisione a 64 bit, reale esteso a 80 bit, intero a 16 bit, intero a 32 bit, intero a 64 bit e intero BCD a 18 cifre), hanno un set di istruzioni che estende quello dell'80286/80386, in quanto include istruzioni di tipo trigonometrico, logaritmico, esponenziale e aritmetico. Inoltre, il coprocessore 80387 supporta un'interfaccia a 32 bit con il bus dati dell'80386, fornisce tutte le funzioni trigonometriche di tangente, seno e coseno (l'80287 supporta solo la funzione tangente per angoli compresi tra 0 e 45 gradi) ed è stato costruito con tecnologia CHMOS III. Il chip 80287 è fornito in un contenitore in ceramica DIP a 40 piedini, mentre il chip 80387 è costruito su matrice in ceramica a 68 piedini (simile a quella utilizzata per l'80286/80386). La frequenza di clock dell'80287 può essere di 5, 8, 10 o 12 MHz, mentre quella dell'80387 può essere di 12 o 16 MHz.
I coprocessori supportano sette tipi di dati, ma internamente lavorano sempre nel formato reale esteso. Le istruzioni di caricamento, che memorizzano i dati sullo stack del coprocessore, convertono automaticamente i dati da uno dei sette tipi al formato reale esteso. Le istruzioni di memorizzazione, che trasferiscono i dati dallo stack del coprocessore alla memoria, eseguono il processo inverso di conversione. La Tabella 9.1 illustra le operazioni realizzate dall'80287/80387.

Tabella 9.1 Operazioni dell'80287/80387

Operazione	**Differenze di comportamento**	
SOMMA intera	Radice quadrata	
SOTTRAZIONE intera	Rapporto di scala (div/molt per potenze di 2)	
MOLTIPLICAZIONE intera	Resto parziale	
DIVISIONE intera	Arrotondamento dell'intero	
CONFRONTO intero	Estrazione mantissa ed esponente	
SOMMA reale	Valore assoluto	
SOTTRAZIONE reale	Cambio segno	
MOLTIPLICAZIONE reale	Test di zero	
DIVISIONE reale	Esame della cima dello stack	
CONFRONTO reale		
Carica zero		
Carica uno		
Carica π		
Carica $\log_2 10$		
Carica $\log_2 e$		
Carica $\text{Log}_{10} 2$		
Carica ln 2		
	80287	**80387**
Tangente	Da 0 a $\pi/4$ rad.	Tutti i valori
Arcotangente	Tutti i valori	Tutti i valori
Seno	Non disponibile	Tutti i valori
Coseno	Non disponibile	Tutti i valori
Seno e coseno simul.	Non disponibile	Tutti i valori
2x − 1	$0 <= x <= .5$	$0 <= x <= .5$
y ($\log_2 x$)	Tutti i valori	Tutti i valori
y ($\log_2(x+1)$)	Tutti i valori	Tutti i valori

Come abbiamo già sottolineato, i coprocessori sono microprocessori che utilizzano lo stack nelle loro elaborazioni. Lo stack può essere immaginato con la forma indicata in Figura 9.1. La cima dello stack (ST: Stack Top) viene referenziata da tre bit contenuti nei registri di lavoro. I numeri possono essere memorizzati nello stack mediante le istruzioni Load e possono essere estratti dallo stack mediante le istruzioni Pop.

ST(0)	ST(stack top) dopo l'inizializzazione
ST(1)	
ST(2)	
ST(3)	
ST(4)	
ST(5)	
ST(6)	
ST(7)	

Figura 9.1 Lo stack dell'80287/80387

9.2 Operazioni sui numeri interi

Sia i microprocessori 80286/80386 che i coprocessori 80287/80387 supportano l'aritmetica sui numeri interi. Esistono tre buone ragioni perché anche i coprocessori siano in grado di manipolare numeri interi:

1. è indispensabile spesso moltiplicare numeri reali con numeri interi;
2. è possibile eseguire più accuratamente e rapidamente le operazioni di elevata precisione su numeri interi che su numeri reali;
3. è possibile eseguire le operazioni tra numeri reali sul coprocessore e restituire in memoria un risultato intero arrotondato (questo è utile in diverse applicazioni grafiche).

Esaminiamo due semplici programmi che realizzano alcune operazioni sui numeri interi.

Somma di due numeri interi
Analizzate dettagliatamente il seguente listato di programma, che sembra non presentare differenze sostanziali con quelli discussi nei precedenti capitoli.

```
;per calcolatori con processori matematici 80287/80387
;programma che esegue una semplice somma tra interi utilizzando
;il coprocessore 80287/80387. I risultati vengono esaminati con
;il programma Debug.

PAGE ,132                          ;dimensionamento pagina

 .8087                             ;direttiva per presenza coprocessore

STACK   SEGMENT PARA STACK
        DB     64 DUP ('MYSTACK ')
STACK   ENDS
```

```
DATI      SEGMENT PARA 'DATI'
NUM1      DQ      123456789ABCDEF0H
NUM2      DD      12345678H
RIS       DQ      ?
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'   ;definisce il segmento di codice
PROCEDURA PROC    FAR             ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS              ;salva DS sullo stack
          SUB     AX,AX           ;azzera AX
          PUSH    AX              ;salva 0 sullo stack
          MOV     AX,DATI         ;indirizzo di DATI in AX
          MOV     DS,AX           ;indirizzo di DATI in DS

          FINIT                   ;inizializza il coprocessore
          FILD    NUM1            ;carica NUM1 in cima allo stack del coprocessore
          FIADD   NUM2            ;somma NUM2 a NUM1 e risultato in cima allo stack
          FISTP   RIS             ;estrae la cima dello stack e memorizza in RIS
          FWAIT                   ;sincronizzazione

          RET                     ;il controllo ritorna al DOS
PROCEDURA ENDP                    ;fine della procedura
CODICE    ENDS                    ;fine del segmento di codice

          END                     ;fine del programma
```

Questo programma contiene una direttiva (.8087) all'assembler che permette di includere il set di istruzioni del coprocessore e di convertire tutti i numeri reali, che seguono la direttiva, dal formato Microsoft al formato IEEE. La struttura dei segmenti dati e stack è quella consueta, ad eccezione dei valori interi particolarmente elevati che sono memorizzati nelle variabili NUM1 e NUM2. Il segmento di codice presenta anch'esso la usuale struttura, ma si differenzia per le seguenti cinque linee di codice:

```
FINIT             ;inizializza il coprocessore
FILD     NUM1     ;carica NUM1 in cima allo stack del coprocessore
FIADD    NUM2     ;somma NUM2 a NUM1 e risultato in cima allo stack
FISTP    RIS      ;estrae la cima dello stack e memorizza in RIS
FWAIT             ;sincronizzazione
```

Si tenga presente che ognuno di questi mnemonici (come qualunque mnemonico di coprocessore) inizia con la lettera F (abbreviazione del termine "fast", cioè rapido). Quando l'80286/80386 esegue il codice macchina corrispondente a questi mnemonici, attiva il coprocessore e passa ad esaminare (in parallelo) l'istruzione successiva.
L'istruzione FINIT esegue l'equivalente di un reset hardware sul chip 80287/80387. In realtà, viene svuotato il contenuto dello stack e azzerati tutti i flag di eccezione e di interruzioni attive. FILD realizza l'operazione di caricamento di un numero intero: l'operando sorgente (NUM1) viene prelevato dalla locazione di memoria specificata e, prima di essere caricato in cima allo stack del coprocessore (cioè in ST(0)), viene convertito nel formato

temporary real (80 bit). Questo operando può essere di uno dei seguenti tre tipi interi: short (32 bit, tipo DD), word (16 bit, tipo DW) oppure long (64 bit, tipo DQ). L'istruzione FIADD somma l'operando sorgente (NUM2) al contenuto corrente del registro che si trova in cima allo stack (ST) e memorizza il risultato in cima allo stack. FIADD supporta due tipi interi: word (DW) e short (DD). Si ricordi che, una volta che un numero è memorizzato nello stack del coprocessore, assume il formato a 80 bit temporary real. FISTP esegue l'estrazione del contenuto del registro che si trova in cima allo stack e lo memorizza in una locazione di memoria specificata (RIS). In questo modo, il coprocessore arrotonda il numero – espresso nel formato temporary real – ad un intero, in accordo con il campo RC della parola di controllo (di cui parleremo dettagliatamente più avanti). Da ultimo, l'istruzione FWAIT sincronizza il coprocessore con il microprocessore 80286/80386 e viene di solito utilizzata prima che venga eseguita nuovamente una istruzione dell'80286/80386. Riportiamo qui di seguito l'immagine del segmento dati dopo l'esecuzione del programma.

```
85B0:0100  1E 2B C0 50 B8 C2 85 8E-D8 9B DF 2E 00 00 9B DA  .+@P8B..X._....Z
85B0:0110  06 08 00 9B DF 3E 0C 00-9B CB 00 00 00 00 00 00  ...._>...K......
85B0:0120  F0 DE BC 9A 78 56 34 12-78 56 34 12 68 35 F1 AC  p^<.xV4.xV4.h5q.
85B0:0130  78 56 34 12 00 00 00 00-00 00 00 00 00 00 00 00  xV4.............
85B0:0140  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
85B0:0150  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
85B0:0160  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
85B0:0170  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20  MYSTACK MYSTACK
```

Una parte di questa immagine di memoria è la seguente:

```
1E 2B C0 50 B8 C2 85 8E-D8 9B DF 2E 00 00 9B DA
06 08 00 9B DF 3E 0C 00-9B CB 00 00 00 00 00 00
F0 DE BC 9A 78 56 34 12-78 56 34 12 68 35 F1 AC
78 56 34 12 00 00 00 00-00 00 00 00 00 00 00 00
```

La sequenza di termini F0 DE BC 9A 78 56 34 12 rappresenta il valore di NUM1, visualizzato in byte, con il byte più significativo in corrispondenza della zona di memoria inferiore e il byte meno significativo in corrispondenza della locazione di memoria superiore. I byte 78 56 34 12 rappresentano il valore di NUM2, mentre 68 35 F1 AC 78 56 34 12 costituiscono la somma esadecimale 12345678ACF13568.
Sicuramente, se non avessimo utilizzato il coprocessore, avremmo dovuto codificare un numero ben maggiore di istruzioni 80286/80386 per raggiungere lo stesso grado di precisione, incrementando inevitabilmente il tempo di esecuzione.
I coprocessori sono definiti microprocessori *stack-oriented*, in quanto tutte le operazioni utilizzano lo stack interno del chip. Al contrario, l'80286/80386 è un microprocessore *memory (register)-oriented*, poiché tutte le operazioni

referenziano un registro e sovente l'informazione viene prelevata dalla memoria o trasferita in memoria.

Somma di una sequenza di numeri interi
Nel prossimo esempio, approfondiamo il problema trattato nel paragrafo precedente, in modo da calcolare la somma di una sequenza di numeri, contenuti in una tabella che è memorizzata in un segmento dati. Riportiamo qui di seguito una parte del segmento dati:

```
NUMERI
          DD  11111111H,22222222H,33333333H,44444444H,55555555H
          DD  66666666H,77777777H,88888888H,99999999H,0AAAAAAAAH
          DD  0BBBBBBBBH,0CCCCCCCCH,0DDDDDDDDH,0EEEEEEEEH
          DD  0FFFFFFFFH,12345678H,9ABCDEF0H
```

Si tenga presente che i numeri referenziati tramite la variabile NUMERI sono di tipo doubleword (DD) – cioè di 32 bit – e che il numero di dimensione maggiore che l'istruzione FIADD può manipolare direttamente è di tipo short integer, cioè di 32 bit.
Il prossimo programma esegue la somma utilizzando un semplice ciclo.

```
;per calcolatori con processori matematici 80287/80387
;programma che esegue la somma di un insieme di numeri interi di
;grandi dimensioni memorizzati nel formato tabellare, utilizzando
;il coprocessore 80287/80387. I risultati vengono esaminati con
;il programma Debug.

PAGE ,132                       ;dimensionamento pagina

 .8087                          ;direttiva per presenza coprocessore

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
NUMERI
          DD  11111111H,22222222H,33333333H,44444444H,55555555H
          DD  66666666H,77777777H,88888888H,99999999H,0AAAAAAAAH
          DD  0BBBBBBBBH,0CCCCCCCCH,0DDDDDDDDH,0EEEEEEEEH
          DD  0FFFFFFFFH,12345678H,9ABCDEF0H
RIS       DQ  ?
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'  ;definisce il segmento di codice
PROCEDURA PROC    FAR            ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS             ;salva DS sullo stack
          SUB     AX,AX          ;azzera AX
          PUSH    AX             ;salva 0 sullo stack
          MOV     AX,DATI        ;indirizzo di DATI in AX
          MOV     DS,AX          ;indirizzo di DATI in DS

          MOV     CX,14          ;somma i primi 15 numeri in NUMERI
```

```
          LEA    BX,NUMERI
          FINIT                ;inizializza il coprocessore
          FILD   NUMERI[BX]    ;estrae il primo numero da NUMERI
ANCORA:   ADD    BX,04H        ;referenzia il prossimo numero di tipo DD
          FIADD  NUMERI[BX]    ;somma con la cima dello stack
          LOOP   ANCORA
          FISTP  RIS           ;salva il risultato contenuto in cima allo stack
          FWAIT                ;sincronizzazione

          RET                  ;il controllo ritorna al DOS
PROCEDURA ENDP                 ;fine della procedura
CODICE    ENDS                 ;fine del segmento di codice

          END                  ;fine del programma
```

Riportiamo qui di seguito una parte del programma:

```
          MOV    CX,14         ;somma i primi 15 numeri in NUMERI
          LEA    BX,NUMERI
          FINIT                ;inizializza il coprocessore
          FILD   NUMERI[BX]    ;estrae il primo numero da NUMERI
ANCORA:   ADD    BX,04H        ;referenzia il prossimo numero di tipo DD
          FIADD  NUMERI[BX]    ;somma con la cima dello stack
          LOOP   ANCORA
          FISTP  RIS           ;salva il risultato contenuto in cima allo
                               ;stack
          FWAIT                ;sincronizzazione
```

In questo esempio, viene eseguita la somma dei primi quindici numeri interi della tabella, utilizzando un ciclo che viene ripetuto un numero di volte pari al numero di somme da effettuare decrementato di una unità (contenuto iniziale del registro CX), in quanto il primo numero viene estratto dalla tabella prima che il programma entri in ciclo. Il registro BX contiene l'offset – riferito all'indirizzo di base di NUMERI – dell'addendo corrente e viene utilizzata l'istruzione LEA (load effective address) per caricare in BX il primo elemento. FINIT inizializza il coprocessore, mentre l'istruzione FILD carica il primo numero intero (nel formato temporary real) in cima allo stack (ST) del coprocessore. Quando il programma entra in ciclo, il contenuto del registro BX viene incrementato di 4 per referenziare così il numero seguente (DD). L'istruzione FIADD NUMERI[BX] somma questo numero al contenuto corrente del registro che si trova in cima allo stack (ST) e memorizza in questo registro il risultato della somma. Questa operazione viene ripetuta fino a quando i 15 numeri sono stati sommati insieme.
Durante questa operazione, la somma parziale è contenuta in cima allo stack del coprocessore e, solo quando tutte le somme sono state eseguite, il risultato finale viene estratto dallo stack e memorizzato nella locazione di memoria referenziata dalla variabile RIS. Dal punto di vista della precisione, è molto utile lasciare i numeri sullo stack del coprocessore fino a quando non risulta indispensabile trasferirli in memoria. In questo modo, infatti,

gli errori di arrotondamento sono minimi. Il programma termina con il comando di sincronizzazione FWAIT.
Utilizziamo ora il programma DEBUG per esaminare il contenuto del segmento dati, dopo l'esecuzione del programma, in modo da verificare il valore del risultato. Riportiamo qui si seguito una parte del segmento dati:

```
7EAC:0100  1E 2B C0 50 88 BF 7E 8E-D8 B9 0E 00 8D 1E 00 00  .+@P8?    .X9......
7EAC:0110  9B DB 87 00 00 83 C3 04-9B DA 87 00 00 E2 F6 9B  .[....C..Z...bv.
7EAC:0120  DF 3E 44 00 9B CB 00 00-00 00 00 00 00 00 00 00  _>D..K..........
7EAC:0130  11 11 11 11 22 22 22 22-33 33 33 33 44 44 44 44  ....""""3333DDDD
7EAC:0140  55 55 55 55 66 66 66 66-77 77 77 77 88 88 88 88  UUUUffffwwww....
7EAC:0150  99 99 99 99 AA AA AA AA-BB BB BB BB CC CC CC CC  ....****;;;;LLLL
7EAC:0160  DD DD DD DD EE EE EE EE-FF FF FF FF 78 56 34 12  ]]]]nnnn....xV4.
7EAC:0170  F0 DE BC 9A F8 FF FF FF-FF FF FF FF 00 00 00 00  p^<.x...........
```

Il risultato della somma viene memorizzato in posizione successiva a quella occupata dall'ultimo numero contenuto nella variabile NUMERI, cioè dopo il valore esadecimale 9ABCDEF0, che corrisponde in memoria alla sequenza F0 DE BC 9A. Questo significa che la stringa di byte F8 FF FF FF FF FF FF FF rappresenta il risultato finale. In particolare, componendo i byte, si ottiene il valore esadecimale FFFFFFFFFFFFFFF8.
Calcolando il risultato a mano, si ottiene invece 7FFFFFFF8, cioè un valore esadecimale che, in apparenza, sembra essere in disaccordo con il valore precedente. In realtà, il coprocessore esegue un'aritmetica con segno su numeri interi, per cui, nel caso di numeri di tipo short integer, qualunque numero uguale o minore di 7FFFFFFF è considerato positivo, mentre qualunque numero superiore a 80000000 è considerato negativo (queste considerazioni possono essere estese anche a numeri interi aventi una dimensione differente). Il coprocessore, in definitiva, ha eseguito una somma algebrica di numeri, di cui gli ultimi otto rappresentano valori negativi. (Facendo questo calcolo a mano, si tenga presente che anche ogni somma parziale superiore a 7FFFFFFF rappresenta un valore negativo).

USO DI MACRO PER VISUALIZZARE NUMERI INTERI

Nei precedenti capitoli, abbiamo scritto le macro per cancellare lo schermo, per posizionare il cursore e per visualizzare alcuni messaggi. La macro che presentiamo ora permette ad un programma di visualizzare direttamente sullo schermo – in corrispondenza della posizione di cursore definita dal registro DX – il contenuto di una variabile di tipo quadword (DQ).

```
COMMENT %Questa macro visualizza sullo schermo una variabile di
tipo DQ (quadword), in corrispondenza della posizione di
cursore definita dal contenuto del registro DX.
Modificando la posizione del cursore e il valore del
puntatore BX, è possibile visualizzare altre variabili.
Questo programma sostituisce il programma DEBUG nelle
```

operazioni di verifica dei programmi 80287/80387.

```
FORMAT:  LEA     BX,VARIABILE       ;locazione della variabile
         MOV     DX,0100H           ;locazione del cursore
         QUAD                       ;chiamata della macro %


QUAD     MACRO                          ;;routine principale
         LOCAL   ANCORA,FINE
         PUSH    DX
         PUSH    CX
         PUSH    BX
         PUSH    AX
         MOV     SI,07H                 ;;valore del contatore
ANCORA:  MOV     AL,BYTE PTR [BX+SI]    ;;un byte degli otto in AL
         MOV     CL,04H                 ;;inizializza CL per rotazione
         ROR     AL,CL                  ;;rotazione del byte di 4 posizioni
         AND     AL,0FH                 ;;solo i 4 bit meno significativi
         POSXY                          ;;sposta il cursore
         VISXY                          ;;visualizza i quattro bit
         MOV     AL,BYTE PTR [BX+SI]    ;;in AL ancora il byte precedente
         AND     AL,0FH                 ;;solo i 4 bit meno significativi
         INC     DX                     ;;incrementa la posizione del cursore
         POSXY                          ;;sposta il cursore
         VISXY                          ;;visualizza quei quattro bit
         INC     DX                     ;;incrementa la posizione del cursore
         DEC     SI                     ;;prossimo byte
         CMP     SI,00H                 ;;gli otto byte sono stati visualizzati?
         JL      FINE                   ;;se sì, fine della macro
         JMP     ANCORA                 ;;altrimenti, esame del byte successivo
FINE:    POP     AX
         POP     BX
         POP     CX
         POP     DX
         ENDM                           ;;fine della macro principale

POSXY    MACRO                          ;;posiziona il cursore sullo schermo
         PUSH    BX                     ;;per visualizzare il byte corrente
         PUSH    AX                     ;;della variabile di tipo DQ
         MOV     BX,01H                 ;;parametri di ingresso
         MOV     AH,02H
         INT     10H
         POP     AX
         POP     BX
         ENDM

VISXY    MACRO                          ;;visualizza numeri e lettere
         LOCAL   CONVER                 ;;esadecimali contenuti in AL
         PUSH    CX
         PUSH    BX
         PUSH    AX
         MOV     AH,10                  ;;parametri per visualizzare caratteri
         MOV     BX,01H
         MOV     CX,01H
         AND     AL,0FH
         CMP     AL,09H
         JLE     CONVERTE               ;;se numero, no lettera
         ADD     AL,07H                 ;;somma 7 al valore del carattere
CONVER:  ADD     AL,30H                 ;;conversione in ASCII
         INT     10H                    ;;chiamata di interruzione
         POP     AX
```

```
POP     BX
POP     CX
ENDM
```

Scrivete e salvate su dischetto questa macro, senza eseguire alcuna operazione di traduzione in codice macchina o di collegamento. In seguito, includetela nel vostro programma. Le istruzioni che occorrono per utilizzare correttamente la macro QUAD sono molto semplici: la locazione della variabile da visualizzare viene trasferita nel registro BX tramite il comando LEA; il registro DX contiene la posizione del cursore da cui si intende iniziare il processo di visualizzazione (DH referenzia la posizione verticale, mentre DL la posizione orizzontale). Nell'esempio, abbiamo supposto che il dato da visualizzare fosse contenuto nel segmento dati del programma e fosse referenziabile tramite l'identificatore VARIABILE. Dunque, le istruzioni che occorrono per visualizzare VARIABILE sullo schermo, in corrispondenza del margine sinistro e della prima linea, sono le seguenti:

```
LEA     BX,VARIABILE
MOV     DX,0100H
QUAD
```

MOLTIPLICAZIONE DI NUMERI INTERI POSITIVI DI GRANDI DIMENSIONI

Il segmento dati del seguente programma contiene tre numeri positivi: 1234H, 5678H e 12345678H. Questi numeri vengono moltiplicati insieme e il risultato viene memorizzato nella variabile RIS di tipo quadword (DQ). Da ultimo, il contenuto di RIS viene visualizzato sullo schermo utilizzando la macro QUAD.

```
;per calcolatori con coprocessori matematici 80287/80387
;programma che esegue la moltiplicazione di un insieme di numeri
;interi di grandi dimensioni. Il risultato viene visualizzato
;sullo schermo utilizzando la macro QUAD.MAC

PAGE ,132                           ;dimensionamento pagina

 .8087                              ;direttiva per presenza coprocessore

IF1
        INCLUDE C:MACLIB.MAC        ;inclusione della libreria MACLIB.MAC
        INCLUDE C:QUAD.MAC          ;inclusione della macro QUAD.MAC per
ENDIF                               ;visualizzare variabili di tipo quadword

STACK   SEGMENT PARA STACK
        DB      48 DUP ('STACK ')
STACK   ENDS

DATI    SEGMENT PARA 'DATI'
NUMERI  DD      1234H,5678H,12345678H
```

```
RIS       DQ      ?
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'  ;definisce il segmento di codice
PROCEDURA PROC    FAR            ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS             ;salva DS sullo stack
          SUB     AX,AX          ;azzera AX
          PUSH    AX             ;salva 0 sullo stack
          MOV     AX,DATI        ;indirizzo di DATI in AX
          MOV     DS,AX          ;indirizzo di DATI in DS

          MOV     CX,2           ;tre numeri da moltiplicare
          LEA     BX,NUMERI      ;indirizzo di NUMERI
          FINIT                  ;inizializza il coprocessore
          FILD    NUMERI[BX]     ;estrae il primo numero
ANCORA:   ADD     BX,04H         ;referenzia il prossimo numero
          FIMUL   NUMERI[BX]     ;moltiplicazione e risultato in ST (stack top)
          LOOP    ANCORA
          FISTP   RIS            ;estrazione di ST e salvataggio in RIS
          FWAIT                  ;sincronizzazione

          CANCELLA               ;cancella lo schermo
          MOV     DX,0100H       ;visualizza il risultato sulla linea 2
          LEA     BX,RIS         ;indirizzo di RIS in BX
          QUAD

          RET                    ;il controllo ritorna al DOS
PROCEDURA ENDP                   ;fine della procedura
CODICE    ENDS                   ;fine del segmento di codice

          END                    ;fine del programma
```

Si noti come il programma includa due librerie esterne di macro tramite la direttiva IF1.

```
IF1
          INCLUDE C:MACLIB.MAC   ;inclusione della libreria MACLIB.MAC
          INCLUDE C:QUAD.MAC     ;inclusione della macro QUAD.MAC per
ENDIF                            ;visualizzare variabili di tipo quadword
```

Nel nostro esempio, MACLIB.MAC permette di cancellare lo schermo, mentre QUAD.MAC visualizza la variabile RIS.
La parte di codice più significativa del programma è semplice e risulta simile a quella del precedente esempio, in cui veniva realizzata l'operazione di somma di alcuni numeri interi:

```
          MOV     CX,2           ;tre numeri da moltiplicare
          LEA     BX,NUMERI      ;indirizzo di NUMERI
          FINIT                  ;inizializza il coprocessore
          FILD    NUMERI[BX]     ;estrae il primo numero
ANCORA:   ADD     BX,04H         ;referenzia il prossimo numero
          FIMUL   NUMERI[BX]     ;moltiplicazione e risultato in ST
                                 ;(stack top)
```

```
LOOP     ANCORA
FISTP    RIS           ;estrazione di ST e salvataggio in RIS
FWAIT                  ;sincronizzazione
```

FIMUL presenta le stesse limitazioni di FIADD per quanto riguarda la dimensione dell'operando sorgente, che deve essere di tipo short-integer (32 bit) oppure di tipo word-integer (16 bit). In questo programma, CX viene inizializzato ancora ad un valore pari al numero complessivo di numeri da moltiplicare diminuito di una unità, in quanto il primo numero viene caricato in cima allo stack del coprocessore prima di entrare in ciclo.
Le istruzioni necessarie per eseguire la visualizzazione del risultato sullo schermo sono le seguenti:

```
CANCELLA                ;cancella lo schermo
MOV       DX,0100H      ;visualizza il risultato sulla linea 2
LEA       BX,RIS        ;indirizzo di RIS in BX
QUAD
```

Dopo l'esecuzione, viene visualizzato sullo schermo il seguente risultato:

```
006FEDD279706D00
```

Questo valore rappresenta il prodotto dei tre numeri nel formato esadecimale.

VISUALIZZAZIONE DI UN GRUPPO DI INTERI SULLO SCHERMO

La macro QUAD può essere chiamata ripetutamente per visualizzare un insieme di numeri. Il prossimo programma, che esegue l'estrazione di radice dei numeri interi compresi tra 0 e 20, illustra come bisogna procedere.

```
;per calcolatori con coprocessori matematici 80287/80387
;programma che illustra come eseguire una semplice operazione
;aritmetica su un numero reale e visualizzare il risultato sullo
;schermo nel formato intero.
;Il programma calcola la radice quadrata dei numeri interi
;compresi tra 1 e 20 e visualizza il risultato sullo schermo
;invocando ripetutamente la macro QUAD. La risposta è nel formato
;esadecimale.

PAGE ,132                        ;dimensionamento pagina

 .8087                           ;direttiva per presenza coprocessore

IF1
         INCLUDE C:MACLIB.MAC    ;inclusione della libreria MACLIB.MAC
         INCLUDE C:QUAD.MAC      ;inclusione della macro QUAD.MAC per
ENDIF                            ;visualizzare variabili di tipo quadword
```

```
STACK     SEGMENT PARA STACK
          DB      48 DUP ('STACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
NUM       DD      1
COST      DQ      100000000
RIS       DQ      ?
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'  ;definisce il segmento di codice
PROCEDURA PROC    FAR            ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS             ;salva DS sullo stack
          SUB     AX,AX          ;azzera AX
          PUSH    AX             ;salva 0 sullo stack
          MOV     AX,DATI        ;indirizzo di DATI in AX
          MOV     DS,AX          ;indirizzo di DATI in DS

          CANCELLA               ;cancella lo schermo
          MOV     DX,0000H       ;posizione del cursore per la prima risposta
          FINIT                  ;inizializza il coprocessore
ANCORA:   FILD    NUM            ;estrae il numero
          FSQRT                  ;esegue l'estrazione di radice del numero
          FILD    COST           ;moltiplica il risultato per COST
          FMUL                   ;moltiplicazione e prodotto in ST (stack top)
          FISTP   RIS            ;estrazione di ST e risultato finale in RIS
          FWAIT                  ;sincronizzazione

          LEA     BX,RIS         ;indirizzo di RIS in BX
          QUAD
          INC     DH             ;sposta il cursore in basso di una linea
          INC     BYTE PTR NUM   ;somma 1 a NUM
          CMP     BYTE PTR NUM,21;esaminati i primi 20 numeri interi?
          JE      FINE           ;se si, fine del programma
          JMP     ANCORA         ;se no, continua

FINE:

          RET                    ;il controllo ritorna al DOS
PROCEDURA ENDP                   ;fine della procedura
CODICE    ENDS                   ;fine del segmento di codice

          END                    ;fine del programma
```

Quando i numeri interi vengono caricati sullo stack del coprocessore, assumono il formato a 80 bit temporary real (questa considerazione vale anche per i numeri reali); inoltre, le istruzioni come FIADD e FIMUL convertono direttamente il formato del numero da intero a reale (80 bit).
In questo programma, un numero intero viene caricato nello stack del coprocessore. L'operazione di estrazione di radice fornisce un numero frazionario (ad esempio, la radice quadrata di 2 è 1.4142...) e, se viene utilizzata l'istruzione FISTP, nella variabile RIS viene memorizzata solo la parte intera (1 nel caso della radice di 2). Per ottenere un risultato intero più preciso, senza ricorrere a routine di conversione di numeri reali, il valore della radice quadrata viene moltiplicato per 100 000 000 e poi estratto dallo stack. Per

utilizzare correttamente la macro QUAD, dovete inserire la virgola decimale nella giusta posizione. Non è difficile, poi, visualizzare sullo schermo un punto, che indichi la virgola decimale, utilizzando le routine presentate nel Capitolo 5. Il segmento dati deve anche contenere le costanti che servono al programma, poiché non è possibile utilizzare istruzioni del coprocessore che abbiano come operandi valori immediati.

```
NUM     DD      1
COST    DQ      100000000
RIS     DQ      ?
```

La prima parte del programma è la seguente:

```
          CANCELLA                ;cancella lo schermo
          MOV       DX,0000H      ;posizione del cursore per la prima
                                  ;risposta
          FINIT                   ;inizializza il coprocessore
ANCORA:   FILD      NUM           ;estrae il numero
          FSQRT                   ;esegue l'estrazione di radice del numero
          FILD      COST          ;moltiplica il risultato per COST
          FMUL                    ;moltiplicazione e prodotto in ST
                                  ;(stack top)
          FISTP     RIS           ;estrazione di ST e risultato finale in RIS
          FWAIT                   ;sincronizzazione
```

La macro CANCELLA viene invocata per cancellare lo schermo, prima di visualizzare i risultati dell'operazione di estrazione di radice. Il registro DX contiene il valore della posizione da cui deve avere inizio la visualizzazione dei risultati sullo schermo. DH referenzia la posizione verticale (da 0 a 24), mentre DL referenzia la posizione orizzontale (da 0 a 64). In questo programma, DL rimane fisso al valore 0, in modo che i risultati vengano visualizzati a partire dalla prima colonna.
Il numero intero NUM viene trasferito in cima allo stack (ST) del coprocessore tramite l'istruzione FILD. Si tenga presente che, una volta memorizzato sullo stack, questo numero assume il formato a 80 bit temporary real. Dopo che è stata eseguita l'estrazione di radice, il risultato viene memorizzato in cima allo stack e si sovrappone al numero precedentemente allocato. Il moltiplicatore COST viene caricato in cima allo stack, spostando il valore della radice quadrata una posizione più in basso nello stack del coprocessore, cioè in ST(1). L'istruzione FMUL esegue la moltiplicazione, utilizzando per difetto come operando sorgente ST(1) e come operando destinazione ST. Se intendete utilizzare come operandi altri registri dello stack, tra gli otto a disposizione, dovete specificarlo:

```
FMUL      ST,ST(6)
```

Il numero reale contenuto in cima allo stack viene estratto e convertito nel formato intero, prima di essere memorizzato nella variabile RIS.
La parte di codice restante, che riportiamo qui di seguito, permette di visualizzare correttamente i risultati sullo schermo e di interrompere al momento opportuno il programma:

```
LEA     BX,RIS              ;indirizzo di RIS in BX
QUAD
INC     DH                  ;sposta il cursore in basso di una linea
INC     BYTE PTR NUM        ;somma 1 a NUM
CMP     BYTE PTR NUM,21     ;esaminati i primi 20 numeri interi?
JE      FINE                ;se sì, fine del programma
JMP     ANCORA              ;se no, continua
```

Si noti come l'indirizzo di RIS venga memorizzato nel registro BX, tramite l'istruzione LEA. La macro QUAD visualizza il risultato in corrispondenza della posizione corrente del cursore. DH (posizione di riga) viene incrementato di una unità per ogni nuovo numero da visualizzare, mentre NUM1 viene incrementato di una unità e confrontato con il valore 21 per verificare se tutti i risultati desiderati sono stati prodotti dal programma. In caso di verifica negativa, viene eseguito ancora il ciclo. Riportiamo qui di seguito l'immagine dei dati prodotti sullo schermo:

```
0000000005F5E100
00000000086DEB2C
000000000A52E659
000000000BEBC200
000000000D53F80E
000000000E999FEE
000000000FC5189B
0000000010DBD658
0000000011E1A300
0000000012D940B6
0000000013C4C48F
0000000014A5CCB2
00000000157DA278
00000000164D50EB
000000001715B41F
0000000017D78400
0000000018935C23
000000001949C185
0000000019FB26E6
000000001AA7F01B
```

I risultati sono nel formato esadecimale, naturalmente. Se l'ultimo numero (000000001AA7F01B) viene convertito nel formato decimale, si ottiene 447213595. Dividendo questo numero per 100 000 000 si ricava la radice quadrata di 20: 4.47213595. È possibile ottenere una maggiore precisione del risultato incrementando la dimensione della costante COST.

9.3 Operazioni su numeri reali e coprocessori Intel

Nei precedenti esempi di questo capitolo, abbiamo allocato in memoria e visualizzato sullo schermo i numeri nel formato intero. Il motivo di questa scelta dipende da come i numeri reali vengono codificati nel coprocessore. Consideriamo questa parte di programma (il resto verrà discusso in un prossimo paragrafo di questo capitolo):

```
PAGE ,132                                   ;dimensionamento pagina
.8087                                       ;direttiva per presenza coprocessore
STACK    SEGMENT    PARA STACK
         DB         48 DUP ('STACK ')
STACK    ENDS
DATI     SEGMENT    PARA 'DATI'
RAGGIO   DQ         8.567
AREA     DQ         ?
DATI     ENDS
```

Se non includiamo la direttiva .8087, il numero reale 8.567 (variabile RAGGIO) viene codificato nel formato Microsoft, cioè nel formato utilizzato dai linguaggi che non supportano il coprocessore. La presenza della direttiva .8087, invece, determina la codifica del numero 8.567 nel formato in virgola mobile IEEE, cioè nel formato richiesto dai coprocessori Intel. Per trasferire e memorizzare numeri utilizzando il processore 80286/80386, i numeri devono essere nel formato intero. Questi numeri vengono automaticamente codificati in un particolare formato quando il programma viene tradotto (in seguito vedremo come ciò avvenga). Sfortunatamente, non esiste un processo di decodifica – analogo a quello svolto dall'assembler – per i numeri che vengono restituiti dal coprocessore. Questo significa che i risultati devono essere riconvertiti dal programmatore, perché risultino comprensibili all'utente. Si tenga presente che:

1. la Microsoft fornisce, nel suo debugger, una opzione per visualizzare i dati nel formato reale. Naturalmente, questo significa che non si può fare a meno del debugger per visualizzare i risultati reali;
2. la IBM fornisce, insieme al suo assembler, diverse routine di conversione collocate in una libreria di utilità chiamata IBMUTIL.LIB. Utilizzando questa libreria, i numeri possono essere decodificati in un formato che sia comprensibile agli utenti. Naturalmente, si deve sempre ricorrere alla libreria IBMUTIL.LIB, ma questo non comporta grandi svantaggi.

Se nessuna di queste soluzioni vi interessa, potete scrivere delle routine di conversione, ma pensateci bene prima di farlo.

FORMATI IEEE PER NUMERI REALI

Esistono diversi modi con cui un numero reale può essere codificato utilizzando cifre esadecimali. Fortunatamente, la IEEE definisce un formato standard in virgola mobile, che la Intel ha adottato per i suoi coprocessori 80287/80387. Sebbene i coprocessori manipolino i dati nel formato temporary real (80 bit), sono i formati short real (32 bit) e long real (64 bit) che ora ci interessa esaminare. Nella definizione delle variabili all'interno di un segmento dati, ad un numero di tipo short real viene riservata una doubleword (DD), mentre ad un numero di tipo long real viene riservata una quadword (DQ).
Il numero di tipo short real viene allocato in memoria con un valore codificato su 32 bit. Il bit più significativo (MSB) viene utilizzato come bit di segno, i successivi otto bit contengono l'esponente, mentre i restanti 23 bit memorizzano la mantissa. Nel caso di numeri di tipo long real, vengono allocati 64 bit. Il bit più significativo viene utilizzato come bit di segno, i successivi undici bit contengono l'esponente, mentre i restanti 52 bit memorizzano la mantissa. L'equazione generale di conversione è abbastanza semplice:

$$numero\ reale = (-1)^{segno} \times (mantissa) \times 2^{esponente}$$

Consideriamo un esempio su un numero di tipo long real. Supponiamo che un programma abbia generato il seguente risultato:

```
82  ED  FC  79  51  D2  6C  40
```

Disponendo i bit nel corretto ordine, si ottiene questo numero reale nel formato esadecimale:

406CD25179FCED82

che, convertito nel formato binario, diventa:

0100000001101100110100100101000101111001111111001110110110000010

Separiamo i bit del numero, in modo da evidenziare il segno, l'esponente e la mantissa:

0 10000000110 1100110100100101000101111001111111001110110110000010

Utilizzando l'equazione generale di conversione, otteniamo i seguenti risultati: il bit più significativo è uguale a 0, per cui -1 elevato a 0 dà come risultato 1, cioè il numero è positivo. La sequenza di bit 10000000110, convertita nel formato decimale, fornisce come risultato 1030. Per numeri di tipo long

real, l'esponente effettivo si ottiene sottraendo al precedente valore il numero 1023: nel nostro caso, l'esponente vale dunque 7. La terza sequenza di bit rappresenta il valore in base 2 della mantissa, che viene preceduta da un 1 e da un punto perché possa essere convertita correttamente nel formato decimale:

1.1100110100100101000101111001111111001110110110000010

La conversione di questo numero binario nel formato decimale obbliga a lavorare con termini frazionari. Per eseguire la conversione di numeri frazionari, è bene procedere con cautela. La prima posizione alla destra della virgola decimale rappresenta 1/2, la seconda 1/4, la terza 1/8, e così via. Ad ogni posizione in cui c'è un 1, viene aggiunto il peso corrispondente a quella cifra:

```
1.0
 .5
 .25
 .03125
 .015625
 .00390625
 .00048828125
 .00006103515625
----------------
1.80133056640625
```

Questo procedimento riguarda solo i primi otto 1. La vostra calcolatrice tascabile probabilmente non può soddisfare la precisione richiesta, ma avrete capito come procedere.
Ora ritorniamo alla formula:

$$\begin{aligned} \textit{numero reale} &= +1.80133056640625 \times 2^7 \\ &= +1.80133056640625 \times 128 \\ &= +230.5703124 \end{aligned}$$

Il risultato è 230.5724458637233.
La conversione di numeri di tipo short real avviene in maniera simile: la mantissa ora è di 23 bit, mentre l'esponente effettivo si calcola convertendo nel formato decimale la sequenza di bit che costituisce il campo esponente del numero e sottraendo al risultato così ottenuto il valore 127. Esaminiamo un altro esempio.
Supponiamo che il seguente numero di tipo short real sia stato prodotto dall'esecuzione di un programma e sia stato memorizzato in un segmento dati:

00 00 58 BE

La versione esadecimale del numero è la seguente:

BE580000

Convertiamo il numero nel formato binario:

10111110010110000000000000000000

Evidenziamo il segno, l'esponente e la mantissa:

1 01111100 10110000000000000000000

Il bit di segno è 1; (−1) elevato a 1 stabilisce che si tratta di un numero negativo. L'esponente 01111100, convertito nel formato decimale, vale 124. L'esponente effettivo viene calcolato sottraendo a 124 il valore 127, per cui si ottiene −3. La mantissa diventa:

1.10110000000000000000000

Convertiamo il numero nel formato decimale:

```
1.0
 .5
 .125
 .0625
------
1.6875
```

Ritornando alla formula, abbiamo:

$$\textit{numero reale} = -1.6875 \times 2^{-3}$$
$$= -1.6875 \times (.125)$$
$$= -.2109375$$

Nei prossimi esempi, utilizzeremo le diverse routine di conversione fornite dalla Microsoft e dalla IBM, in sostituzione della conversione manuale.

SEMPLICE PROGRAMMA DI ARITMETICA SU NUMERI REALI

Questo esempio utilizza il programma della Microsoft Symbolic Debug per visualizzare il contenuto del segmento dati del programma, che calcola l'area di un cerchio utilizzando l'equazione $A = \pi R^2$. Il raggio del cerchio e il valore dell'area vengono restituiti nel formato reale (precisione a 64 bit).

```
;per calcolatori con coprocessori matematici 80287/80387
;il programma esegue una semplice operazione aritmetica su numeri
```

```
;reali e il risultato reale viene visualizzato utilizzando il
;programma Symbolic Debugger della Microsoft.
;Il programma calcola l'area di un cerchio

PAGE ,132                          ;dimensionamento pagina

 .8087                             ;direttiva per presenza coprocessore

STACK     SEGMENT PARA STACK
          DB      64 DUP ('MYSTACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
RAGGIO    DQ      8.567
AREA      DQ      ?
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'  ;definisce il segmento di codice
PROCEDURA PROC    FAR              ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK
          PUSH    DS               ;salva DS sullo stack
          SUB     AX,AX            ;azzera AX
          PUSH    AX               ;salva 0 sullo stack
          MOV     AX,DATI          ;indirizzo di DATI in AX
          MOV     DS,AX            ;indirizzo di DATI in DS

          FINIT                    ;inizializza il coprocessore
          FLD     RAGGIO           ;valore di RAGGIO in cima allo stack
          FMUL    RAGGIO           ;il quadrato di RAGGIO in cima allo stack
          FLDPI                    ;π in cima allo stack
          FMUL                     ;πR**2
          FSTP    AREA             ;estrazione dallo stack e salvataggio in AREA
          FWAIT                    ;sincronizzazione

          RET                      ;il controllo ritorna al DOS
PROCEDURA ENDP                     ;fine della procedura
CODICE    ENDS                     ;fine del segmento di codice

          END                      ;fine del programma
```

I numeri reali possono essere memorizzati nel segmento dati con il formato DD o DQ. Il tipo doubleword (DD) viene utilizzato per i numeri reali di 32 bit, mentre il tipo quadword (DQ) viene utilizzato per i numeri reali di 64 bit. I numeri reali possono avere i seguenti formati:

```
   8.567
 +12.345
-234.9
 -12.4E2
   1.234E-10
```

Riportiamo qui di seguito una parte del segmento di codice:

```
FINIT              ;inizializza il coprocessore
FLD       RAGGIO   ;valore di RAGGIO in cima allo stack
FMUL      RAGGIO   ;il quadrato di RAGGIO in cima allo stack
```

```
FLDPI              ;π in cima allo stack
FMUL               ;πR**2
FSTP     AREA      ;estrazione dallo stack e salvataggio in AREA
FWAIT              ;sincronizzazione
```

L'istruzione FLD viene utilizzata per trasferire il numero reale dalla memoria al registro che si trova in cima dello stack del coprocessore, mentre FMUL moltiplica questo numero per il valore della variabile RAGGIO, che è nel formato reale. Questa istruzione memorizza il quadrato del raggio in cima allo stack. FLDPI carica il valore π in cima allo stack, spostando R^2 in ST(1). L'istruzione FMUL moltiplica ST per ST(1) e memorizza il risultato in cima allo stack. L'istruzione finale, FSTP, trasferisce il risultato, espresso nel formato reale, nella variabile AREA.
A questo punto utilizziamo il programma Symbolic Debug della Microsoft, per esaminare il contenuto del segmento dati. Riportiamo qui di seguito la stessa zona di memoria in due forme diverse:

```
-D DS:0100 014F
85B0:0100  1E 2B C0 50 B8 C2 85 8E-D8 9B DD 06 00 00 9B DC   .+@P8B..X.]....\
85B0:0110  0E 00 00 9B D9 EB 9B DE-C9 9B DD 1E 08 00 9B CB   ....Yk.^I.]....K
85B0:0120  FC A9 F1 D2 4D 22 21 40-82 ED FC 79 51 D2 6C 40   :)qRM"!@.m:yQR1@
85B0:0130  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20   MYSTACK MYSTACK
85B0:0140  4D 59 53 54 41 43 4B 20-4D 59 53 54 41 43 4B 20   MYSTACK MYSTACK

-DL DS:0100 014F
85B0:0100  1E 2B C0 50 B8 C2 85 8E  -0.1044298876043245E+65299
85B0:0108  D8 9B DD 06 00 00 9B DC  -0.1255977334619014E-65397
85B0:0110  0E 00 00 9B D9 EB 9B DE  -0.5578447610515987E-65388
85B0:0118  C9 9B DD 1E 08 00 9B CB  -0.1655102779899069E+57
85B0:0120  FC A9 F1 D2 4D 22 21 40  +0.8567E+1
85B0:0128  82 ED FC 79 51 D2 6C 40  +0.2305724458637233E+3
85B0:0130  4D 59 53 54 41 43 4B 20  +0.4066692443677049E+65384
85B0:0138  4D 59 53 54 41 43 4B 20  +0.4066692443677049E+65384
85B0:0140  4D 59 53 54 41 43 4B 20  +0.4066692443677049E+65384
```

La prima immagine della memoria – da 85B0:0100 a 85B0:014F – è nella forma normale (prodotta dal programma Debug della IBM). La seconda immagine della memoria viene visualizzata tramite l'opzione DL (Dump Long):

```
-DL DS:0100 014F
```

I numeri reali vengono ricavati da gruppi di otto byte. Alcuni di questi numeri, come accade in tutte le visualizzazioni di memoria, sono privi di significato. La coppia di numeri che a noi interessa è la seguente:

```
85B0:0120  FC  A9  F1  D2  4D  22  21  40  +0.8567E+1
85B0:0128  82  ED  FC  79  51  D2  6C  40  +0.2305724458637233E+3
```

Il primo numero (8.567) rappresenta il valore del raggio, mentre il secondo numero (230.5724458637233) è il valore dell'area del cerchio che è stata calcolata dal programma.

ROUTINE DI CONVERSIONE DATI PER MACRO ASSEMBLER IBM

Insieme al Macro Assembler IBM – a partire dalla versione 2.0 – vengono fornite anche alcune routine di conversione dati. Queste routine permettono al programmatore di convertire le stringhe ASCII nel formato in virgola mobile 80287/80387, di convertire i numeri, espressi nel formato in virgola mobile long real, in caratteri ASCII e di convertire i numeri, espressi nel formato Microsoft, nel formato 80287/80387 e viceversa. Le opzioni a disposizione del programmatore sono le seguenti:

$I8_INPUT	da stringhe ASCII a 80287/80387
$I8_OUTPUT	da long real 80287/80387 a stringhe ASCII
$I4_M4	da Microsoft singola a short real 80287/80387
$I8_M8	da Microsoft doppia a long real 80287/80387
$M4_I4	da short real 80287/80387 a Microsoft in singola precisione
$M8_I8	da long real 80287/80387 a Microsoft in doppia precisione
$I4_I8	da long real 80287/80387 a short real 80287/80387
$I8_I4	da short real 80287/80387 a long real 80287/80387

Queste routine di conversione sono particolarmente utili per interfacciare il codice assemblatore con un linguaggio di alto livello che non supporta i coprocessori. La libreria delle routine di conversione (IBMUTIL), che contiene tutte queste opzioni, viene collegata al programma principale utilizzando il Linker IBM.
Il nostro programma invoca solo una tra queste routine di conversione, dovendo convertire i numeri reali 80287/80387 nella notazione scientifica (in virgola mobile). La routine $I8_OUTPUT prende il numero espresso nel formato reale e lo converte in una stringa di caratteri ASCII che può essere visualizzata direttamente sullo schermo.
Quando si utilizza la routine di conversione $I8_OUTPUT devono essere rispettate le seguenti condizioni:

1. Il segmento di codice deve essere dichiarato in questo modo:

```
CODMAT  SEGMENT  BYTE      PUBLIC   'CODE'
        EXTRN    $I8_OUTPUT:NEAR
```

   (Qui di seguito si inserisce il contenuto del segmento di codice)

```
CODMAT  ENDS
```

2. Il segmento dati viene dichiarato in questo modo:

```
DATI    SEGMENT  WORD      PUBLIC   'DATI'
```

   (Qui di seguito si inserisce il contenuto del segmento di dati)

```
DATA    ENDS
DGROUP  GROUP    DATI
```

3. Tutte le routine sono procedure di tipo NEAR.

4. I registri DS e ES sono uguali.

5. Il contenuto di tutti i registri che non sono registri di segmento (ad eccezione di SP) viene alterato.

6. SI:DS punta all'indirizzo (offset) del numero di tipo long real prima che la routine venga chiamata.

7. Dopo l'esecuzione della routine, SI:DS punta all'indirizzo (offset) di LSTRING. Questa è una locazione di memoria di 17 byte che contiene, nel primo byte, la lunghezza della stringa ASCII prodotta. La virgola decimale è a sinistra.

8. AX contiene il valore 1 se è stato utilizzato il numero originale, e il valore 0 se il numero da convertire era indefinito.

9. BL contiene un carattere spazio nel caso di numero positivo e una lineetta nel caso di numero negativo.

10. DX contiene l'esponente del numero espresso nel formato in base 10.

```
B>C:MASM NOMEPROG
IBM Personal Computer MACRO Assembler   Version 2.00
(C)Copyright IBM Corp 1981, 1984
(C)Copyright Microsoft Corp 1981, 1983, 1984


48774 Bytes free

Warning Severe
Errors  Errors
0       0

B>C:LINK NOMEPROG,NOMEPROG,NUL,C:IBMUTIL;

IBM Personal Computer Linker


Version 2.30 (C) Copyright IBM Corp. 1981, 1985

B>
```

Figura 9.2 Tipiche operazioni di traduzione e collegamento in presenza di librerie

L'uscita prodotta dalla routine $I8_OUTPUT non è ancora in un formato piacevole, per cui il programma deve convertirla nella notazione scientifica. La Figura 9.2 mostra una tipica operazione di traduzione e di collegamento tra moduli distinti quando viene utilizzata una libreria. L'assembler e il Linker IBM sono contenuti nel drive C quando la traduzione ha inizio, mentre il programma si trova nel drive B.
Il comando LINK specifica anche il nome della libreria (C:IBMUTIL) che deve essere collegata al programma. Questa libreria viene così inclusa nel programma nel formato eseguibile .EXE.

USO DELLA LIBRERIA DI UTILITÀ IBM

La routine IBM di conversione $I8_OUTPUT converte in una stringa ASCII un numero reale, espresso nel formato in virgola mobile in doppia precisione (DQ). Il registro BL contiene l'indicatore di segno e il registro DX contiene l'esponente del numero in base 10. Il numero da visualizzare deve essere formattato, per cui il seguente listato di programma contiene una parte di codice che converte il numero nella notazione scientifica.

```
;per calcolatori con coprocessori matematici 80287/80387
;il programma esegue una semplice operazione aritmetica su numeri
;reali e il risultato reale viene visualizzato utilizzando la
;routine di conversione dati della IBM.
;IBMUTIL.LIB deve essere inclusa al programma in fase di
;esecuzione.
;Il programma calcola il volume di una sfera.

PAGE ,132                       ;dimensionamento pagina

 .8087                          ;direttiva per presenza coprocessore

IF
        INCLUDE C:MACLIB.MAC
ENDIF

STACK   SEGMENT PARA STACK
        DB      64 DUP ('MYSTACK ')
STACK   ENDS

DATI    SEGMENT PARA 'DATI'     ;il segmento dati deve essere public
COST1   DD      4.0             ;costante numerica
COST2   DD      3.0             ;costante numerica
RAGGIO  DQ      123.45E2        ;raggio della sfera
VOLUME  DQ      ?               ;volume nel formato reale Intel
BLANK   DB      20 DUP (' ')    ;stringa di spazi per la routine IBM
RIS     DB      17 DUP (?),'$'  ;risposta in caratteri dalla routine IBM
PRIMA   DB      ' ','$'         ;locazione per la prima cifra
POT     DW      ?               ;locazione per l'esponente
BUFF    DB      4 DUP (' ')     ;4 byte per i caratteri dell'esponente
SEGNO   DB      '-$'            ;segno negativo
VIRGOLA DB      '.$'            ;virgola decimale
ESP     DB      ' E $'          ;simbolo dell'esponente
DATI    ENDS
```

```
CODMAT    SEGMENT BYTE PUBLIC 'CODE'    ;definisce il segmento di codice per la routine IBM
          EXTRN   $I8_OUTPUT:NEAR       ;routine nella libreria esterna
PROCEDURA PROC    FAR             ;inizio della procedura
          ASSUME  CS:CODMAT,DS:DATI,SS:STACK,ES:DATI
          PUSH    DS              ;salva DS sullo stack
          SUB     AX,AX           ;azzera AX
          PUSH    AX              ;salva 0 sullo stack
          MOV     AX,DATI         ;indirizzo di DATI in AX
          MOV     DS,AX           ;indirizzo di DATI in DS
          MOV     ES,AX           ;indirizzo di DATI in ES

          LEA     SI,VOLUME       ;indirizzo di VOLUME in SI (indice sorgente per la stringa)

          FINIT                   ;inizializza il coprocessore
          FLD     RAGGIO          ;RAGGIO in cima allo stack
          FMUL    RAGGIO          ;in cima allo stack il quadrato di RAGGIO
          FMUL    RAGGIO          ;in cima allo stack il cubo di RAGGIO
          FLDPI                   ;π in cima allo stack
          FMUL                    ;π per (RAGGIO al cubo)
          FMUL    COST1           ;moltiplica per 4.0
          FDIV    COST2           ;divide per 3.0
          FSTP    QWORD PTR [SI]  ;estrazione del risultato e salvataggio in VOLUME
          FWAIT                   ;sincronizzazione

          CALL    $I8_OUTPUT      ;chiamata della routine
          CALL    FORMAT          ;notazione scientifica del risultato

          RET                     ;il controllo ritorna al DOS
PROCEDURA ENDP                    ;fine della procedura

COMMENT / La seguente procedura di tipo NEAR viene utilizzata per
convertire l'uscita della routine $I8_OUTPUT nella
notazione scientifica. Esempio: -3.5678912345 E -123  /

FORMAT    PROC    NEAR
          SUB     DX,01H          ;riduce l'esponente di 1
          MOV     POT,DX          ;DX è l'esponente corretto di RIS
          CMP     BL,'-'          ;verifica se il risultato è + o -
          JNE     NONEG
          LEA     DX,SEGNO        ;se è negativo, visualizza il carattere
          MOV     AH,09
          INT     21H

NONEG:

          CLD                     ;inizio trasferimento stringa in RIS
          MOV     CL,17           ;la stringa è lunga 17 byte (16 cifre)
          LEA     DI,RIS          ;destinazione del trasferimento
REP       MOVSB                   ;trasferimento
          MOV     CL,1            ;trasferisce il primo caratere di RIS in PRIMA
          LEA     SI,RIS[1]
          LEA     DI,PRIMA
REP       MOVSB
          LEA     DX,PRIMA        ;visualizza la prima cifra
          MOV     AH,09
          INT     21H
          LEA     DX,VIRGOLA      ;visualizza la virgola decimale
          MOV     AH,09
          INT     21H
          LEA     DX,RIS[2]       ;visualizza le restanti cifre di RIS
          MOV     AH,09
```

```
        INT     21H
        LEA     DX,ESP          ;visualizza E, cioè il simbolo di esponente
        MOV     AH,09           ;dopo la stringa di caratteri
        INT     21H
        MOV     DX,POT          ;converte il numero in DX in una stringa
        CMP     DX,8000H        ;è positivo o negativo?
        JB      POSIT           ;se positivo, salta a POSIT
        LEA     DX,SEGNO        ;se negativo, visualizza il segno -
        MOV     AH,09
        INT     21H
        MOV     DX,POT          ;valore esadecimale corretto, per un numero negativo
        XOR     DX,OFFFFH
        ADD     DX,01
POSIT:  MOV     CX,0
        LEA     DI,BUFF         ;BUFF viene utilizzata come memoria tampone
POT1:   PUSH    CX              ;di quattro byte quando i numeri esadecimali
        MOV     AX,DX           ;in DX vengono convertiti nei valori decimali
        MOV     DX,0            ;ASCII per essere visualizzati sullo schermo
        MOV     CX,10
        DIV     CX
        XCHG    AX,DX
        ADD     AL,30H          ;conversione in ASCII
        MOV     [DI],AL         ;salvataggio in BUFF
        INC     DI              ;punta alla locazione corrente in BUFF
        POP     CX
        INC     CX              ;CX contiene il numero di cifre
        CMP     DX,0
        JNZ     POT1
SCHERMO: DEC    DI              ;visualizzazione di numeri sullo schermo
        MOV     AL,[DI]         ;prende la cifra da BUFF
        PUSH    DX              ;salva il valore originale di DX
        MOV     DL,AL           ;trasferimento della cifra da visualizzare
        MOV     AH,2            ;parametro per visualizzazione DOS
        INT     21H             ;visualizzazione
        POP     DX              ;ripristino valore originale di DX
        LOOP    SCHERMO         ;continua fino alla fine
        RET
FORMAT  ENDP
CODMAT  ENDS                    ;fine del segmento di codice CODMAT

        END                     ;fine del programma
```

Questo programma calcola il volume di una sfera, utilizzando la formula $V = 4/3\ \pi\ R^3$, e visualizza sullo schermo il risultato nella notazione scientifica. I dati contenuti nel segmento DATI si possono dividere in due categorie: i dati utilizzati per i calcoli e i dati utilizzati per formattare il risultato. I dati appartenenti alla prima categoria sono i seguenti:

```
COST1    DD   4.0        ;costante numerica
COST2    DD   3.0        ;costante numerica
RAGGIO   DQ   123.45E2   ;raggio della sfera
VOLUME   DQ   ?          ;volume nel formato reale Intel
```

Non è possibile passare esplicitamente gli operandi al coprocessore, ma questi devono essere stati allocati prima in memoria. COST1 e COST2 rappresentano rispettivamente i valori 4 e 3 che vengono divisi nell'equazione di

calcolo del volume della sfera. Queste costanti numeriche potevano essere memorizzate come valori interi e utilizzate in quella forma. Il valore del RAGGIO della sfera è 12345. VOLUME è stata definita come variabile di tipo quadword in quanto viene sottoposta al processo di conversione.
La parte restante del segmento dati viene utilizzata per convertire il risultato RIS.

```
BLANK     DB   20 DUP (' ')     ;stringa di spazi per la routine IBM
RIS       DB   17 DUP (?),'$'   ;risposta in caratteri dalla routine IBM
PRIMA     DB   ' ','$'          ;locazione per la prima cifra
POT       DW   ?                ;locazione per l'esponente
BUFF      DB   4 DUP (' ')      ;4 byte per i caratteri dell'esponente
SEGNO     DB   '-$'             ;segno negativo
VIRGOLA   DB   '.$'             ;virgola decimale
ESP       DB   ' E $'           ;simbolo dell'esponente
```

Quando la routine $I8_OUTPUT viene invocata dal programma, il numero puntato dal registro SI (in questo caso, VOLUME) viene convertito in una stringa e il risultato di questa conversione viene memorizzato nella locazione di memoria puntata da SI. Occorre definire un buffer (BLANK) – in questo caso, costituito da 20 caratteri spazio – che separi questa stringa dalle altre variabili all'interno del segmento dati. Alla variabile RIS è stata riservata una zona di memoria di 17 byte in modo che possa contenere stringhe aventi al massimo questa dimensione. Il carattere $ indica che viene invocata l'interruzione INT 21H per visualizzare il risultato sullo schermo. PRIMA memorizza la prima cifra del risultato. Questo programma converte i risultati nella notazione scientifica, per cui vengono visualizzati rispettivamente la prima cifra, la virgola decimale e le restanti cifre del risultato. La variabile POT memorizza l'esponente del risultato, che viene restituito al registro DX dopo l'invocazione della routine $I8_OUTPUT. La variabile BUFF viene utilizzata durante la conversione del valore numerico dell'esponente, contenuto in DX, nel corrispondente carattere ASCII. Da ultimo, SEGNO, VIRGOLA e ESP memorizzano rispettivamente i simboli meno, virgola decimale ed esponente.
Il codice di questo programma si differenzia da quello del precedente programma, in quanto è conforme alle specifiche definite dalla routine $I8_OUTPUT.

```
CODMAT      SEGMENT  BYTE PUBLIC 'CODE'  ;definisce il segmento di
                                         ;codice per la routine IBM
            EXTRN    $I8_OUTPUT:NEAR     ;routine nella libreria
                                         ;esterna
PROCEDURA   PROC     FAR                 ;inizio della procedura
            ASSUME   CS:CODMAT,DS:DATI,SS:STACK,ES:DATI
```

Il segmento di codice CODMAT e la routine $I8_OUTPUT devono essere dichiarati rispettivamente PUBLIC e EXTRN perché possa avvenire uno scam-

bio corretto di informazione con la libreria IBMUTIL.LIB. Questa libreria viene inclusa al programma indicandone il nome completo in fase di collegamento.
Le istruzioni per il coprocessore, che riportiamo qui di seguito, sono molto concise:

```
FLD     RAGGIO    ;RAGGIO in cima allo stack
FMUL    RAGGIO    ;in cima allo stack il quadrato di RAGGIO
FMUL    RAGGIO    ;in cima allo stack il cubo di RAGGIO
```

Il raggio viene prima elevato al cubo e il risultato di questa operazione viene memorizzato in cima allo stack.

```
FLDPI             ;π in cima allo stack
FMUL              ;π per (RAGGIO al cubo)
```

L'istruzione FLDPI carica il valore π in cima allo stack e sposta il cubo del raggio nel registro ST(1). FMUL moltiplica il contenuto di ST(1) con ST e memorizza il prodotto in ST, cioè in cima allo stack.

```
FMUL    COST1             ;moltiplica per 4.0
FDIV    COST2             ;divide per 3.0
FSTP    QWORD PTR [SI]    ;estrazione del risultato e salvataggio in VOLUME
```

Il risultato che si trova in cima allo stack viene moltiplicato per 4 e diviso per 3. L'istruzione FSTP memorizza nella variabile VOLUME il valore finale nel formato reale.

```
CALL    $I8_OUTPUT        ;chiamata della routine
CALL    FORMAT            ;notazione scientifica del risultato
```

L'invocazione della routine $I8_OUTPUT permette di convertire il numero di tipo DQ puntato dal SI (in questo caso, VOLUME) in una stringa, che viene memorizzata nella locazione di memoria originale. La procedura FORMAT preleva l'informazione restituita dalla routine IBM e la formatta nella notazione scientifica. Questa procedura visualizza il risultato nel formato ASCII a partire dalla posizione corrente del cursore. Il processo di formattazione può essere diviso nei seguenti passi:

1. Se il numero è negativo, visualizzare il segno meno
2. Visualizzare la prima cifra del risultato
3. Visualizzare la virgola decimale
4. Visualizzare le restanti cifre del risultato
5. Visualizzare il simbolo E di esponente
6. Se l'esponente è negativo, visualizzare il segno meno
7. Visualizzare l'esponente.

Le seguenti istruzioni stabiliscono se deve essere visualizzato il segno meno (–):

```
FORMAT  PROC    NEAR
        SUB     DX,01H        ;riduce l'esponente di 1
        MOV     POT,DX        ;DX è l'esponente corretto di RIS
        CMP     BL,'–'        ;verifica se il risultato è + o –
        JNE     NONEG
        LEA     DX,SEGNO      ;se è negativo, visualizza il carattere
        MOV     AH,09
        INT     21H
NONEG:
```

Le istruzioni SUB DX,01H e MOV POT,DX salvano l'informazione che riguarda l'esponente. La verifica sul segno viene basata sul contenuto del registro BL. Se BL memorizza il carattere meno (–), il contenuto della variabile SEGNO viene visualizzato sullo schermo. Se, invece, BL contiene qualunque altro carattere (numero positivo), non viene visualizzato nulla sullo schermo. Le 17 cifre ASCII contenute in VOLUME vengono quindi copiate in RIS, utilizzando le seguenti istruzioni:

```
        CLD                   ;inizio trasferimento stringa in RIS
        MOV     CL,17         ;la stringa è lunga 17 byte (16 cifre)
        LEA     DI,RIS        ;destinazione del trasferimento
REP     MOVSB                 ;trasferimento
```

Si tenga presente che SI (registro indice sorgente) punta al contenuto della locazione VOLUME.
La prima cifra del risultato viene ora copiata da RIS a PRIMA.

```
        MOV     CL,1          ;trasferisce il primo carattere di RIS in PRIMA
        LEA     SI,RIS[1]
        LEA     DI,PRIMA
REP     MOVSB
```

Il registro CL viene inizializzato a 1, poiché deve essere copiato un solo carattere, quello memorizzato in RIS[1] e non in RIS[0]. RIS[0], infatti, contiene il numero di caratteri che compongono la stringa.
La seguente parte di codice visualizza sullo schermo la prima cifra (PRIMA), la virgola decimale e i restanti caratteri ASCII della stringa:

```
        LEA     DX,PRIMA      ;visualizza la prima cifra
        MOV     AH,09
        INT     21H
        LEA     DX,VIRGOLA    ;visualizza la virgola decimale
        MOV     AH,09
        INT     21H
```

```
            LEA     DX,RIS[2]    ;visualizza le restanti cifre di RIS
            MOV     AH,09
            INT     21H
```

Questa costituisce una semplice applicazione delle tecniche di visualizzazione di caratteri che abbiamo presentato nel Capitolo 5.
A questo punto, è necessario gestire l'informazione relativa all'esponente. Non c'è difficoltà per quanto riguarda la visualizzazione del simbolo di esponente:

```
            LEA     DX,ESP       ;visualizza E, cioè il simbolo di esponente
            MOV     AH,09        ;dopo la stringa di caratteri
            INT     21H
```

Prima che venga visualizzato l'esponente, questo deve essere convertito nel formato ASCII. Inoltre, si deve stabilire il segno dell'esponente.

```
            MOV     DX,POT       ;converte il numero in DX in una stringa
            CMP     DX,8000H     ;è positivo o negativo?
            JB      POSIT        ;se positivo, salta a POSIT
            LEA     DX,SEGNO     ;se negativo, visualizza il segno –
            MOV     AH,09
            INT     21H
            MOV     DX,POT       ;valore esadecimale corretto, per un numero
                                 ;negativo
            XOR     DX,0FFFFH
            ADD     DX,01
POSIT:      MOV     CX,0
```

POT è la variabile che contiene l'informazione relativa all'esponente in forma decimale. Se POT è minore di 8000H, l'esponente è positivo e viene eseguito un salto all'etichetta POSIT. Se POT è uguale a o maggiore di 8000H, il numero è negativo e viene visualizzato sullo schermo un segno meno. Viene poi calcolato il complemento a due del numero negativo, in modo da convertire l'informazione in un esponente positivo.
Successivamente, vengono convertite le singole cifre di POT.

```
            LEA     DI,BUFF      ;BUFF viene utilizzata come memoria
                                 ;tampone
POT1:       PUSH    CX           ;di quattro byte quando i numeri esadecimali
            MOV     AX,DX        ;in DX vengono convertiti nei valori decimali
            MOV     DX,0         ;ASCII per essere visualizzati sullo schermo
            MOV     CX,10
            DIV     CX
            XCHG    AX,DX
            ADD     AL,30H       ;conversione in ASCII
            MOV     [DI],AL      ;salvataggio in BUFF
            INC     DI           ;punta alla locazione corrente in BUFF
```

```
POP     CX
INC     CX              ;CX contiene il numero di cifre
CMP     DX,0
JNZ     POT1
```

L'esponente contenuto in POT può essere un numero di quattro cifre. Per capire come la routine di conversione operi, supponiamo che l'esponente sia 34. L'istruzione MOV AX,DX pone il valore 34 nel registro AX. Viene poi memorizzato in DX il valore 0, mentre CX contiene il numero 10 che costituisce il divisore nella successiva operazione di divisione. Dopo che questa operazione (DIV CX,AX) è stata eseguita, AX contiene il valore 3 e DX contiene il valore 4. L'istruzione XCHG inverte quest'ordine, per cui AX contiene 4 e DX 3 (in realtà il numero 4 è memorizzato in AL). Quando viene sommato ad AL il valore 30H, si verifica la conversione nel formato ASCII e il carattere così ottenuto viene memorizzato in BUFF come cifra più significativa. Queste operazioni di conversione vengono ripetute anche per il numero 3, che viene così salvato nella locazione corrente di BUFF come carattere ASCII. Il processo di conversione si interrompe, poiché DX contiene il valore 0 quando viene completata l'ultima sequenza di istruzioni DIV e XCHG. Si noti come le cifre in BUFF siano in ordine invertito, ma questo non è un problema.
Da ultimo, viene visualizzato sullo schermo il carattere corrisponente all'esponente:

```
SCHERMO:  DEC   DI        ;visualizzazione di numeri sullo schermo
          MOV   AL,[DI]   ;prende la cifra da BUFF
          PUSH  DX        ;salva il valore originale di DX
          MOV   DL,AL     ;trasferimento della cifra da visualizzare
          MOV   AH,2      ;parametro per visualizzazione DOS
          INT   21H       ;visualizzazione
          POP   DX        ;ripristino valore originale di DX
          LOOP  SCHERMO   ;continua fino alla fine
```

Poiché i caratteri sono stati salvati in ordine invertito, quando vengono visualizzati sullo schermo, vengono letti in ordine posizionale decrescente. Il numero di cifre da visualizzare viene controllato dal contenuto di CX.
Abbiamo discusso le istruzioni che sono necessarie per visualizzare sullo schermo un numero nel formato scientifico. Se pensate che questa procedura sia utile e se state usando il Macro Assembler IBM, potete inserirla nella libreria IBMUTIL.LIB.

CALCOLO DELLA TANGENTE DI UN ANGOLO

I coprocessori Intel supportano la funzione tangente. L'80287 restituisce il risultato corretto quando l'argomento è compreso tra 0 e $\pi/4$ radianti, men-

tre l'80387 calcola la tangente di un angolo qualsiasi. Come anticipazione di applicazioni più complicate che utilizzano funzioni trigonometriche, discutiamo il seguente programma che calcola la tangente di un angolo reale (compreso tra 0 e $\pi/4$) e visualizza sullo schermo il risultato con l'aiuto della libreria IBM.
L'angolo viene specificato in gradi e poi convertito in radianti dal programma.

```
;per calcolatori con coprocessori matematici 80287/80387
;il programma esegue una semplice operazione aritmetica su numeri
;reali e il risultato reale viene visualizzato utilizzando la
;routine di conversione dati della IBM.
;IBMUTIL.LIB deve essere inclusa al programma in fase di
;esecuzione.
;Il programma calcola la tangente di un angolo.
;L'angolo deve essere compreso tra 0 e 45 gradi!

PAGE ,132                           ;dimensionamento pagina

 .8087                              ;direttiva per presenza coprocessore

IF1
         INCLUDE C:MACLIB.MAC
ENDIF

STACK    SEGMENT PARA STACK
         DB      64 DUP ('MYSTACK ')
STACK    ENDS

DATI     SEGMENT PARA 'DATI'    ;il segmento dati deve essere Public
ANGOLO   DD      25.5           ;angolo in gradi
COST     DD      180.0          ;costante numerica
TANGENTE DQ      ?              ;tangente nel formato reale Intel
BLANK    DB      20 DUP (' ')   ;stringa di spazi per la routine IBM
RIS      DB      17 DUP (?),'$' ;risposta in caratteri dalla routine IBM
PRIMA    DB      ' ','$'        ;locazione per la prima cifra
POT      DW      ?              ;locazione per l'esponente
BUFF     DB      4 DUP (' ')    ;4 byte per i caratteri dell'esponente
SEGNO    DB      '-$'           ;segno negativo
VIRGOLA  DB      '.$'           ;virgola decimale
ESP      DB      ' E $'         ;simbolo dell'esponente
DATI     ENDS

CODMAT   SEGMENT BYTE PUBLIC 'CODE'    ;definisce il segmento di codice per la routine IBM
         EXTRN   $I8_OUTPUT:NEAR       ;routine nella libreria esterna
PROCEDURA PROC   FAR            ;inizio della procedura
         ASSUME  CS:CODMAT,DS:DATI,SS:STACK,ES:DATI
         PUSH    DS             ;salva DS sullo stack
         SUB     AX,AX          ;azzera AX
         PUSH    AX             ;salva 0 sullo stack
         MOV     AX,DATI        ;indirizzo di DATI in AX
         MOV     DS,AX          ;indirizzo di DATI in DS
         MOV     ES,AX          ;indirizzo di DATI in ES

         LEA     SI,TANGENTE    ;indirizzo di TANGENTE in SI (indice sorgente per la stringa)

         FINIT                  ;inizializza il coprocessore
         FLDPI                  ;π in cima allo stack
         FLD     COST           ;COST in cima allo stack
         FDIV                   ;divisione di COST per π
```

```
        FLD     ANGOLO          ;ANGOLO in cima allo stack
        FMUL                    ;ANGOLO moltiplicato per il precedente risultato
        FPTAN                   ;calcolo della tangente del prodotto precedente
        FDIV                    ;divisione ST per ST(1)
        FSTP    QWORD PTR [SI]  ;estrazione del risultato e salvataggio in VOLUME
        FWAIT                   ;sincronizzazione

        CALL    $I8_OUTPUT      ;chiamata della routine
        CALL    FORMAT          ;notazione scientifica del risultato

        RET                     ;il controllo ritorna al DOS
PROCEDURA ENDP                  ;fine della procedura

COMMENT / La seguente procedura di tipo NEAR viene utilizzata per
convertire l'uscita della routine $I8_OUTPUT nella
notazione scientifica. Esempio: -3.5678912345 E -123  /

FORMAT  PROC    NEAR
        SUB     DX,01H          ;riduce l'esponente di 1
        MOV     POT,DX          ;DX è l'esponente corretto di RIS
        CMP     BL,'-'          ;verifica se il risultato è + o -
        JNE     NONEG
        LEA     DX,SEGNO        ;se è negativo, visualizza il carattere
        MOV     AH,09
        INT     21H

NONEG:

        CLD                     ;inizio trasferimento stringa in RIS
        MOV     CL,17           ;la stringa è lunga 17 byte (16 cifre)
        LEA     DI,RIS          ;destinazione del trasferimento
REP     MOVSB                   ;trasferimento
        MOV     CL,1            ;trasferisce il primo caratere di RIS in PRIMA
        LEA     SI,RIS[1]
        LEA     DI,PRIMA
REP     MOVSB
        LEA     DX,PRIMA        ;visualizza la prima cifra
        MOV     AH,09
        INT     21H
        LEA     DX,VIRGOLA      ;visualizza la virgola decimale
        MOV     AH,09
        INT     21H
        LEA     DX,RIS[2]       ;visualizza le restanti cifre di RIS
        MOV     AH,09
        INT     21H
        LEA     DX,ESP          ;visualizza E, cioè il simbolo di esponente
        MOV     AH,09           ;dopo la stringa di caratteri
        INT     21H
        MOV     DX,POT          ;converte il numero in DX in una stringa
        CMP     DX,8000H        ;è positivo o negativo?
        JB      POSIT           ;se positivo, salta a POSIT
        LEA     DX,SEGNO        ;se negativo, visualizza il segno -
        MOV     AH,09
        INT     21H
        MOV     DX,POT          ;valore esadecimale corretto, per un numero negativo
        XOR     DX,0FFFFH
        ADD     DX,01
POSIT:  MOV     CX,0
        LEA     DI,BUFF         ;BUFF viene utilizzata come memoria tampone
POT1:   PUSH    CX              ;di quattro byte quando i numeri esadecimali
        MOV     AX,DX           ;in DX vengono convertiti nei valori decimali
```

```
          MOV     DX,0          ;ASCII per essere visualizzati sullo schermo
          MOV     CX,10
          DIV     CX
          XCHG    AX,DX
          ADD     AL,30H        ;conversione in ASCII
          MOV     [DI],AL       ;salavataggio in BUFF
          INC     DI            ;punta alla locazione corrente in BUFF
          POP     CX
          INC     CX            ;CX contiene il numero di cifre
          CMP     DX,0
          JNZ     POT1
SCHERMO:  DEC     DI            ;visualizzazione di numeri sullo schermo
          MOV     AL,[DI]       ;prende la cifra da BUFF
          PUSH    DX            ;salva il valore originale di DX
          MOV     DL,AL         ;trasferimento della cifra da visualizzare
          MOV     AH,2          ;parametro per visualizzazione DOS
          INT     21H           ;visualizzazione
          POP     DX            ;ripristino valore originale di DX
          LOOP    SCHERMO       ;continua fino alla fine
          RET
FORMAT    ENDP
CODMAT    ENDS                  ;fine del segmento di codice CODMAT

          END                   ;fine del programma
```

Il segmento dati contiene due numeri reali: ANGOLO rappresenta il valore dell'angolo in gradi (25.5, in questo caso), mentre COST viene utilizzata per convertire l'angolo in radianti. La struttura del programma è simile a quella del programma precedente. Riportiamo qui di seguito il codice che permette di calcolare la tangente dell'angolo:

```
FINIT                       ;inizializza il coprocessore
FLDPI                       ;π in cima allo stack
FLD     COST                ;COST in cima allo stack
FDIV                        ;divisione di COST per π
FLD     ANGOLO              ;ANGOLO in cima allo stack
FMUL                        ;ANGOLO moltiplicato per il precedente
                            ;risultato
FPTAN                       ;calcolo della tangente del prodotto precedente
FDIV                        ;divisione ST per ST(1)
FSTP    QWORD PTR [SI]      ;estrazione del risultato e salvataggio
                            ;in VOLUME
FWAIT                       ;sincronizzazione
```

π viene caricato in cima allo stack e poi viene memorizzato in ST(1), quando COST viene posta in ST(0). FDIV divide π per 180 e lascia il risultato in cima allo stack. FLD carica il valore di ANGOLO in cima allo stack, inserendo in posizione inferiore sullo stack il risultato della precedente operazione. L'istruzione FMUL moltiplica ST con ST(1) e memorizza il risultato in cima allo stack. Questo valore corrisponde all'angolo in radianti. Il comando FPTAN calcola la tangente dell'angolo come rapporto di due numeri: Y e X. Y viene posto in ST e poi viene inserito in posizione inferiore sullo stack quando X viene memorizzato in cima allo stack. FDIV divide Y per X e memorizza il

quoziente in cima allo stack. L'istruzione FSTP restituisce il risultato nel formato reale e lo memorizza in TANGENTE. Successivamente viene invocata la routine IBM per convertire questo risultato nella notazione scientifica.

ROUTINE CHE CALCOLA IL SENO DI UN ANGOLO

Il coprocessore 80287 non supporta le funzioni seno e coseno, ma, potendo calcolare la funzione tangente per angoli inferiori a 45 gradi ($\pi/4$), si possono ricavare i valori del seno e coseno per angoli di valore qualsiasi. Al contrario, il coprocessore 80387 supporta le funzioni tangente, seno e coseno. Esaminiamo il seguente programma che calcola il seno di un angolo.

```
;per calcolatori con coprocessori matematici 80287/80387
;il programma esegue una semplice operazione aritmetica e
;visualizza il risultato finale con il Symbolic Debugger.
;Il programma calcola il seno di un angolo intero, compreso tra 0
;e 90 gradi.

PAGE ,132                          ;dimensionamento pagina

 .8087                             ;direttiva per presenza coprocessore

STACK     SEGMENT PARA STACK
          DB      48 DUP ('STACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
ANGOLO    DW      65
TEMP      DW      ?
COST      DD      180.0
SENO      DQ      ?
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'  ;definisce il segmento di codice
PROCEDURA PROC    FAR              ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK,ES:DATI
          PUSH    DS               ;salva DS sullo stack
          SUB     AX,AX            ;azzera AX
          PUSH    AX               ;salva 0 sullo stack
          MOV     AX,DATI          ;indirizzo di DATI in AX
          MOV     DS,AX            ;indirizzo di DATI in DS
{*************************************************************}
          MOV     AX,ANGOLO        ;verifica del valore dell'angolo
          CMP     AX,45            ;il valore limite è 45
          JG      AGGIUSTA         ;se è maggiore, salta
          JMP     CONT             ;altrimenti, continua
AGGIUSTA: NEG     AX               ;sottrazione 90-AX
          ADD     AX,90            ;risultato nel registro AX
CONT:     MOV     TEMP,AX          ;salva il risultato in TEMP

          FINIT                    ;inizializza il coprocessore
          FLDPI                    ;π sullo stack
          FLD     COST             ;COST sullo stack
          FDIV                     ;divisione COST per π
          FILD    TEMP             ;carica l'angolo
```

```
        FMUL                    ;moltiplicazione
        FPTAN                   ;calcolo della tangente del prodotto
        FWAIT

        MOV     AX,ANGOLO       ;verifica la dimensione dell'angolo
        CMP     AX,45           ;se è 45 o minore, formula per il seno
        JG      COSX            ;altrimenti, formula per il coseno
        JMP     SENX

COSX:   FXCH    ST(1)           ;aggiorna lo stack per coseno
SENX:   FMUL    ST(0),ST        ;nessun aggiornamento stack per seno
        FXCH    ST(1)
        FLD     ST(0)           ;identità trig. per seno o coseno
        FMUL    ST(0),ST        ;X o Y diviso per l'ipotenusa
        FADD    ST(0),ST(2)
        FSQRT
        FDIVP   ST(1),ST
        FSTP    SENO            ;risultato in SENO
        FWAIT                   ;sincronizzazione
{************************************************************}
        RET                     ;il controllo ritorna al DOS
PROCEDURA ENDP                  ;fine della procedura
CODICE  ENDS                    ;fine del segmento di codice

        END                     ;fine del programma
```

Se si usa il coprocessore 80387, tutte le istruzioni comprese tra le due linee di asterischi possono essere sostituite con il seguente codice semplificato:

```
FINIT
FLDPI
FLD     COST
FDIV
FILD    ANGOLO
FMUL
FSIN
FSTP    SENO
FWAIT
```

Se l'angolo è compreso tra 0 e $\pi/2$ – cioè tra 0 e 90 gradi – il seno dell'angolo coincide con il coseno di $\pi/2$ meno l'angolo:

$$\sin(x) = \cos(\pi/2 - x)$$

Si tenga presente che se l'angolo misura $\pi/4$ (45 gradi), sia il seno che il coseno valgono .707. Queste relazioni vengono utilizzate in modo opportuno dal programma.
L'esempio che abbiamo presentato si compone di tre parti: una che converte l'angolo in radianti e che lo riduce ad un valore uguale o minore di $\pi/4$; una che calcola la tangente dell'angolo così modificato e una che calcola il risultato finale utilizzando la formula di conversione del coseno, se il valore originale dell'angolo era maggiore di $\pi/4$, oppure la formula di conversione del seno, se il valore originale dell'angolo era inferiore a $\pi/4$.

Le istruzioni che riducono il valore dell'angolo ad una misura compresa tra 0 e $\pi/4$ sono le seguenti:

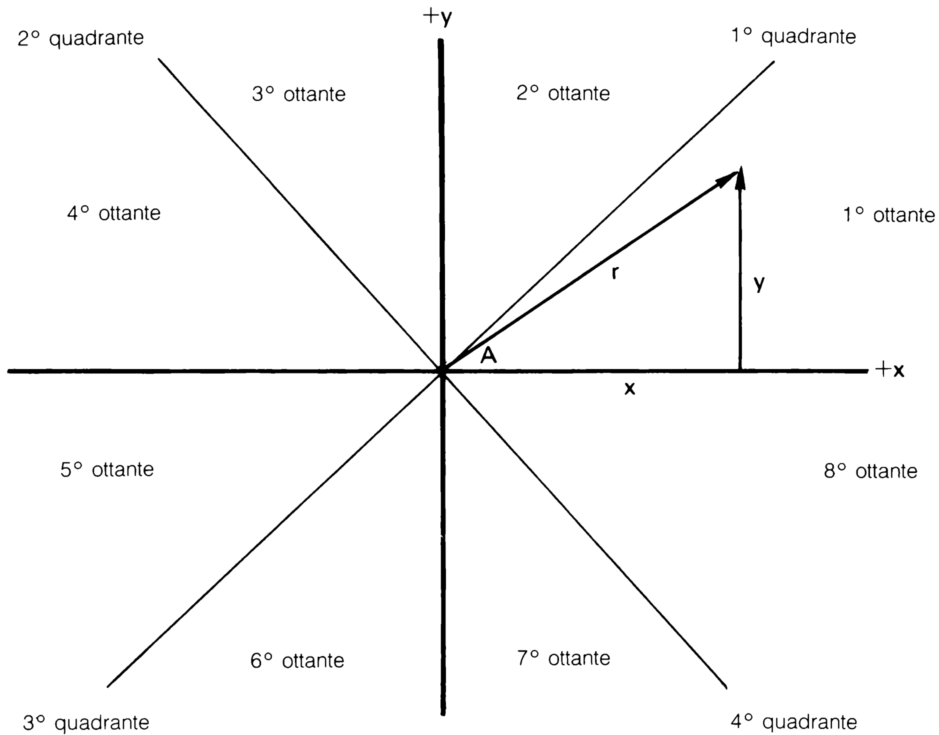
```
            MOV    AX,ANGOLO     ;verifica del valore dell'angolo
            CMP    AX,45         ;il valore limite è 45
            JG     AGGIUSTA      ;se è maggiore, salta
            JMP    CONT          ;altrimenti, continua
AGGIUSTA:   NEG    AX            ;sottrazione 90 – AX
            ADD    AX,90         ;risultato nel registro AX
CONT:       MOV    TEMP,AX       ;salva il risultato in TEMP
```

La variabile ANGOLO è di tipo word integer (DW) e contiene il valore originale dell'angolo in gradi. Se quest'angolo è maggiore di 45 gradi, il programma salta alla etichetta AGGIUSTA; altrimenti, viene eseguita l'istruzione JMP all'etichetta CONT. Nel primo caso, il contenuto di AX viene complementato e gli viene sommato 90 (gradi), cioè viene sottratto a 90 il valore dell'angolo. Il risultato viene memorizzato nella variabile TEMP, per essere utilizzato in seguito dalla funzione tangente.

```
FINIT                ;inizializza il coprocessore
FLDPI                ;π sullo stack
FLD      COST        ;COST sullo stack
FDIV                 ;divisione COST per π
FILD     TEMP        ;carica l'angolo
FMUL                 ;moltiplicazione
FPTAN                ;calcolo della tangente del prodotto
FWAIT
```

Queste istruzioni sono simili a quelle dell'esempio precedente, con l'eccezione che TEMP contiene ora un valore intero. Quando viene eseguita l'istruzione FPTAN, i valori X e Y da dividere vengono memorizzati rispettivamente in cima allo stack (ST) e una locazione sotto la cima dello stack.
Aiutatevi con il triangolo in Figura 9.3 quando viene spiegato come calcolare il seno dell'angolo.
Il coprocessore ha memorizzato nello stack i due valori X e Y, che costituiscono il risultato della precedente operazione. Resta dunque da calcolare R per ottenere la funzione seno o coseno.

```
(1) COSX:   FXCH    ST(1)         ;aggiorna lo stack per coseno
(2) SENX:   FMUL    ST(0),ST      ;nessun aggiornamento stack per seno
(3)         FXCH    ST(1)
(4)         FLD     ST(0)         ;identità trig. per seno o coseno
(5)         FMUL    ST(0),ST      ;X o Y diviso per l'ipotenusa
(6)         FADD    ST(0),ST(2)
(7)         FSQRT
(8)         FDIVP   ST(1),ST
(9)         FSTP    SENO          ;risultato in SENO
(10)        FWAIT                 ;sincronizzazione
```



$$R = \sqrt{X^2 + Y^2}$$

1° quadrante: $\sin = \cos(\pi/4 - \theta)$

$$\sin\theta = \frac{Y}{R} = \frac{Y}{\sqrt{X^2 + Y^2}}$$

$$\cos\theta = \frac{X}{R} = \frac{X}{\sqrt{X^2 + Y^2}}$$

Figura 9.3 Informazioni trigonometriche per il programma che utilizza la serie di Fourier

Esaminiamo la Tabella 9.2 per renderci conto di come venga modificato il contenuto dello stack quando viene eseguita questa parte di programma. Se il valore originale dell'angolo non è superiore a $\pi/4$ (45 gradi), si applica la formula del calcolo del seno, mentre se è maggiore di $\pi/4$ (e minore o uguale a $\pi/2$, cioè 90 gradi) si applica la formula di calcolo del coseno ($\cos(x) = \sin(90-x)$). Quindi, per calcolare il seno di un angolo, è possibile utilizzare angoli i cui valori siano maggiore anche di $\pi/4$. Al passo (2), in cima allo stack si trova X^2 o Y^2. Analizzando la sequenza di operazioni elencate in Tabella 9.2, si può notare come $R = (X^2 + Y^2)$ venga collocato in cima allo stack. Al passo (8) R viene diviso per Y (nel caso del seno) o per X (nel caso del coseno) e il risultato viene memorizzato in ST(1), che diventa il nuovo ST. Prima che il passo venga completato, comunque, viene estratto il contenuto del registro che si trova in cima allo stack e viene trasferito il quoziente in ST(0). Quando viene eseguito il passo (9), il risultato effettivo viene memorizzato nella variabile SENO e corretto per ogni angolo compreso tra 0 e $\pi/2$.
I risultati forniti dal programma possono essere analizzati con il programma Symbolic Debug della Microsoft. Riportiamo qui di seguito una parte del segmento dati:

```
46FB:0158  F9  9B  DD  1E  08  00  9B  CB   -0.1655102779799079E+57
46FB:0160  41  00  19  00  00  00  34  43   +0.5629499535851585E+16
46FB:0168  67  D7  2D  30  79  00  ED  3F   +0.9063077870366499E+0
46FB:0170  4D  59  53  54  41  43  4B  20   +0.4066692443677049E-152
```

Il seno di 65 gradi è, con notevole precisione, +0.9063077870366499E+0. Sebbene quella che abbiamo presentato sia una semplice applicazione di tri-

Tabella 9.2 Attività dello stack durante l'esecuzione del programma che calcola il seno di un angolo

	Calcolo del seno			Calcolo del coseno		
	ST(0)	ST(1)	ST(2)	ST(0)	ST(1)	ST(2)
(0)	X	Y		X	Y	
(1)				Y	X	
(2)	X^2	Y		Y^2	X	
(3)	Y	X^2		X	Y^2	
(4)	Y	Y	X^2	X	X	Y^2
(5)	Y^2	Y	X^2	X^2	X	Y^2
(6)	Y^2+X^2	Y	X^2	X^2+Y^2	X	Y^2
(7)	$(Y^2+X^2)^{.5}$	Y	X^2	$(X^2+Y^2)^{.5}$	X	Y^2
(8)	R*	Y/R	X^2	R	X/R	Y^2
(9)	R	Seno	X^2	R	Coseno	Y^2

$*R = \sqrt{x^2+y^2}$

gonometria e algebra, sicuramente il programma viene notevolmente semplificato utilizzando le istruzioni dell'80387.

DEFINIZIONE DI UNA TABELLA DI VALORI DEL SENO A PRECISIONE ELEVATA

Apportando alcune modifiche al programma precedente, è possibile definire una Tabella di conversione che contenga il valore della funzione seno per angoli compresi tra 0 e 90 gradi. Il seguente listato di programma indica le modifiche che sono state apportate:

```
;per calcolatori con coprocessori matematici 80287/80387
;il programma esegue una semplice operazione aritmetica e viene
;analizzato il risultato finale con il programma Symbolic
;Debugger.
;Il programma calcola il seno di un angolo intero, compreso tra 0
;e 90 gradi, e memorizza il risultato in memoria.

PAGE ,132                               ;dimensionamento pagina

 .8087                                  ;direttiva per presenza coprocessore

STACK     SEGMENT PARA STACK
          DB      48 DUP ('STACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
ANGOLO    DW      0                     ;valore originale dell'angolo
TEMP      DW      ?                     ;locazione di memoria temporanea
COST      DD      180.0                 ;costante per conversione
SENO      DQ      91 DUP (?)            ;locazione per il risultato
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'  ;definisce il segmento di codice
PROCEDURA PROC    FAR                   ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK,ES:DATI
          PUSH    DS                    ;salva DS sullo stack
          SUB     AX,AX                 ;azzera AX
          PUSH    AX                    ;salva 0 sullo stack
          MOV     AX,DATI               ;indirizzo di DATI in AX
          MOV     DS,AX                 ;indirizzo di DATI in DS

          LEA     BX,SENO
ANCORA:   MOV     AX,ANGOLO             ;verifica del valore dell'angolo
          CMP     AX,45                 ;il valore limite è 45
          JG      AGGIUSTA              ;se è maggiore, salto
          JMP     CONT                  ;altrimenti, continua
AGGIUSTA: NEG     AX                    ;sottrazione 90-AX
          ADD     AX,90                 ;risultato nel registro AX
CONT:     MOV     TEMP,AX               ;salva il risultato in TEMP

          FINIT                         ;inizializza il coprocessore
          FLDPI                         ;π sullo stack
          FLD     COST                  ;COST sullo stack
          FDIV                          ;divisione COST per π
          FILD    TEMP                  ;carica l'angolo
```

```
          FMUL                      ;moltiplicazione
          FPTAN                     ;calcolo della tangente del prodotto
          FWAIT

          MOV     AX,ANGOLO         ;verifica la dimensione dell'angolo
          CMP     AX,45             ;se è 45 o minore, formula per il seno
          JG      COSX              ;altrimenti, formula per il coseno
          JMP     SENX

COSX:     FXCH    ST(1)             ;aggiorna lo stack per coseno
SENX:     FMUL    ST(0),ST          ;nessun aggiornamento stack per seno
          FXCH    ST(1)
          FLD     ST(0)             ;identità trig. per seno o coseno
          FMUL    ST(0),ST          ;X o Y diviso per l'ipotenusa
          FADD    ST(0),ST(2)
          FSQRT
          FDIVP   ST(1),ST
          FSTP    SENO[BX]          ;risultato in SENO
          FWAIT                     ;sincronizzazione

          ADD     BX,08H            ;punta alla successiva locazione in SENO
          ADD     ANGOLO,1          ;incrementa l'angolo
          CMP     ANGOLO,90         ;verifica se l'angolo è di 91 gradi
          JLE     ANCORA            ;se no, ripete il calcolo

          RET                       ;il controllo ritorna al DOS
PROCEDURA ENDP                      ;fine della procedura
CODICE    ENDS                      ;fine del segmento di codice

          END                       ;fine del programma
```

Si noti come solo la seguente parte di codice è stata modificata:

```
FSTP     SENO[BX]        ;risultato in SENO
FWAIT                    ;sincronizzazione
ADD      BX,08H          ;punta alla successiva locazione in SENO
ADD      ANGOLO,1        ;incrementa l'angolo
CMP      ANGOLO,90       ;verifica se l'angolo è di 91 gradi
JLE      ANCORA          ;se no, ripete il calcolo
```

Nel segmento dati, la variabile SENO è stata definita come una zona di memoria contenente 91 valori di tipo DQ (defined quadword). Per ogni funzione seno che viene calcolata, BX viene incrementato di otto byte in modo da puntare alla successiva locazione libera nella tabella SENO. La variabile ANGOLO viene confrontata con il valore 90 (gradi) e, in base al risultato di questa verifica, si decide se terminare il programma o se continuare nell'esecuzione. La Tabella 9.3 elenca i valori delle funzioni seno prodotti dal programma.

Tabella 9.3 Valori a precisione elevata del seno per angoli compresi tra 0 e 90

+0.0E+0
+0.1745240643728351E)−1
+0.3489949670250097E)−1
+0.5233595624294383E)−1
+0.697564737441253E)−1
+0.8715574274765818E)1
+0.1045284632676535E+0
+0.1218693434051475E+0
+0.139173100960654E+0
+0.1564344650402309E+0
+0.1736481776669304E+0
+0.1908089953765448E+0
+0.2079116908177593E+0
+0.224951054343865E+0
+0.2419218955996677E+0
+0.2588190451025207E+0
+0.2756373558169992E+0
+0.2923717047227367E+0
+0.3090169943749475E+0
+0.3255681544571566E+0
+0.3420201433256687E+0
+0.3583679495453003E+0
+0.374606593415912E+0
+0.3907311284892738E+0
+0.4067366430758002E+0
+0.4226182617406994E+0
+0.4383711467890774E+0
+0.4539904997395468E+0
+0.4694715627858908E+0
+0.484809620246337E+0
+0.5E+0
+0.5150380749100542E+0
+0.5299192642332049E+0
+0.5446390350150271E+0
+0.5591929034707468E+0
+0.573576436351046E+0
+0.5877852522924731E+0
+0.6018150231520483E+0
+0.6156614753256583E+0
+0.6293203910498375E+0
+0.6427876096865394E+0
+0.6560590289905073E+0
+0.6691306063588582E+0

Tabella 9.3 (continua)

+0.6819983600624985E+0
+0.6946583704589973E+0
+0.7071067811865476E+0
+0.7193398003386512E+0
+0.7313537016191705E+0
+0.7431448254773942E+0
+0.754709580222772E+0
+0.766044443118978E+0
+0.7771459614569709E+0
+0.7880107536067219E+0
+0.7986355100472928E+0
+0.8090169943749475E+0
+0.8191520442889918E+0
+0.8290375725550417E+0
+0.8386705679454241E+0
+0.848048096156426E+0
+0.8571673007021123E+0
+0.8660254037844386E+0
+0.8746197071393959E+0
+0.882947592858927E+0
+0.8910065241883679E+0
+0.898794046299167E+0
+0.9063077870366499E+0
+0.9135454576426009E+0
+0.9205048534524404E+0
+0.9271838545667874E+0
+0.9335804264972017E+0
+0.9396926207859084E+0
+0.9455185755993168E+0
+0.9510565162951535E+0
+0.9563047559630354E+0
+0.9612616959383189E+0
+0.9659258262890683E+0
+0.9702957262759965E+0
+0.9743700647852352E+0
+0.9781476007338057E+0
+0.981627183447664E+0
+0.984807753012208E+0
+0.9876883405951378E+0
+0.9902680687415704E+0
+0.992546151641322E+0
+0.9945218953682733E+0

Tabella 9.3 (continua)

+0.9961946980917455E+0
+0.9975640502598242E+0
+0.9986295347545738E+0
+0.9993908270190958E+0
+0.9998476951563913E+0
+0.1E+1

VISUALIZZAZIONE DI UNA FORMA D'ONDA SINUSOIDALE

Il programma che abbiamo presentato nel paragrafo precedente calcolava il seno di angoli compresi tra 0 e 90 gradi e restituiva un risultato reale in SENO[BX]. Per visualizzare questi risultati sullo schermo in forma grafica, è necessario apportare alcune modifiche al programma originale. I risultati reali devono essere convertiti in valori interi e rapportati alle dimensioni dello schermo. Inoltre, devono essere calcolati i valori delle coordinate dell'onda sinusoidale, per ogni angolo compreso tra 91 e 360 gradi.
I passi necessari per convertire un numero reale nel formato intero si riducono a modificare solamente l'istruzione FSTP SENO con l'istruzione FISTP SENO, dopo aver adattato il valore delle funzioni seno alle dimensioni dello schermo. Lo schermo ad alta risoluzione del PC AT IBM si compone verticalmente di 200 pixel e orizzontalmente di 640 pixel. Un'onda sinusoidale varia tra +1.00 e −1.00. Si assume quindi come fattore di scala il valore 100, in modo da poter visualizzare un'onda sinusoidale tra i valori +100 e −100 (ampiezza picco picco di 200) nella direzione verticale. Se la posizione orizzontale viene incrementata di un'unità (pixel) per ogni grado dell'angolo, occorrono solo 360 pixel, dei 640 a disposizione. Per centrare l'onda sullo schermo, viene allora utilizzato un offset di 140 pixel a partire dal margine sinistro (640−360=280 e 280/2=140). La parte di codice che esegue i due passi appena descritti è la seguente:

```
FIMUL   MOLTIP
FISTP   SENO[SI]     ;risultato in memoria
```

dove MOLTIP vale 100. Il registro SI viene utilizzato come indice per spaziare all'interno della tabella SENO. Si verifica, comunque, una notevole perdita di precisione nei risultati, in quanto si ottiene una precisione non superiore alla terza cifra significativa, ma questa limitazione è dovuta al tipo di risoluzione dello schermo e non al calcolatore.

Il passo 3 consiste nel visualizzare i punti dell'onda sinusoidale per angoli compresi tra 91 e 360 gradi. Ci sono due strade da seguire: si può cambiare l'algoritmo, per calcolare tutti i 361 valori, oppure si può ripetere quattro volte il procedimento che è stato utilizzato per gli angoli compresi tra 0 e 90 gradi.
Si tenga presente che la seconda strada è stata seguita nel Capitolo 8, quando abbiamo visualizzato sullo schermo un'onda sinusoidale prelevando i valori da una tabella di conversione. La principale differenza esistente tra questo programma e quello del Capitolo 8 è che ora i valori devono essere calcolati e non sono già presenti in una tabella. Il programma che ora esaminiamo ricorda quello che avevamo presentato nel Capitolo 8 (Figura 8.1), ma utilizza anche le tecniche illustrate negli ultimi due esempi appena discussi.

```
;per calcolatori con coprocessori matematici 80287/80387
;il programma esegue una semplice operazione aritmetica su numeri
;reali e, dopo averli ridotti in scala, li converte in numeri
;interi.
;Il programma calcola il seno di  angoli interi, compresi tra 0
;e 90 gradi, e visualizza l'onda sinusoidale (da 0 a 360 gradi)
;sullo schermo grafico

PAGE ,132                           ;dimensionamento pagina

 .8087                              ;direttiva per presenza coprocessore

SCHERMO  MACRO                      ;;inizializza lo schermo ad alta
         MOV    AH,00               ;;risoluzione in b/n 200 per 640
         MOV    AL,06
         INT    10H
         ENDM

DISPLAY  MACRO                      ;;macro per visualizzare i punti
         MOV    AH,12
         MOV    AL,01
         MOV    CX,POS
         ADD    CX,140              ;;immagine al centro dello schermo
         MOV    DH,00
         MOV    DL,BASSO
         INT    10H
         ENDM

STACK    SEGMENT PARA STACK
         DB     48 DUP ('STACK ')
STACK    ENDS

DATI     SEGMENT PARA 'DATI'
ANGOLO   DW     0                   ;valore originale dell'angolo
TEMP     DW     ?                   ;locazione di memoria temporanea
COST     DD     180.0               ;costante per conversione
MOLTIP   DW     100                 ;converte il seno da 0 a 100
POS      DW     0
BASSO    DB     0
SENO     DQ     91 DUP (?)          ;locazione per il risultato
DATI     ENDS
```

```
CODICE    SEGMENT PARA 'CODICE'  ;definisce il segmento di codice
PROCEDURA PROC    FAR            ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK,ES:DATI
          PUSH    DS             ;salva DS sullo stack
          SUB     AX,AX          ;azzera AX
          PUSH    AX             ;salva 0 sullo stack
          MOV     AX,DATI        ;indirizzo di DATI in AX
          MOV     DS,AX          ;indirizzo di DATI in DS
          MOV     SI,0           ;SI punta ad una locazione di SENO
ANCORA:   MOV     AX,ANGOLO      ;verifica del valore dell'angolo
          CMP     AX,45          ;il valore limite è 45
          JG      AGGIUSTA       ;se è maggiore, salto
          JMP     CONT           ;altrimenti, continua
AGGIUSTA: NEG     AX             ;sottrazione 90-AX
          ADD     AX,90          ;risultato nel registro AX
CONT:     MOV     TEMP,AX        ;salva il risultato in TEMP

          FINIT                  ;inizializza il coprocessore
          FLDPI                  ;π sullo stack
          FLD     COST           ;COST sullo stack
          FDIV                   ;divisione COST per π
          FILD    TEMP           ;carica l'angolo
          FMUL                   ;moltiplicazione
          FPTAN                  ;calcolo della tangente del prodotto
          FWAIT

          MOV     AX,ANGOLO      ;verifica la dimensione dell'angolo
          CMP     AX,45          ;se è 45 o minore, usa il seno
          JG      COSX           ;altrimenti, usa il coseno
          JMP     SENX

COSX:     FXCH    ST(1)          ;aggiorna lo stack per coseno
SENX:     FMUL    ST(0),ST       ;nessun aggiornamento stack per seno
          FXCH    ST(1)
          FLD     ST(0)          ;identità trig. per seno o coseno
          FMUL    ST(0),ST       ;X o Y diviso per l'ipotenusa
          FADD    ST(0),ST(2)
          FSQRT
          FDIVP   ST(1),ST
          FIMUL   MOLTIP
          FSTP    SENO[BX]       ;risultato in SENO
          FWAIT                  ;sincronizzazione

          ADD     BX,08H         ;punta alla successiva locazione in SENO
          ADD     ANGOLO,1       ;incrementa l'angolo
          CMP     ANGOLO,90      ;verifica se l'angolo è di 91 gradi
          JLE     ANCORA         ;se no, ripete il calcolo

;routine simile a qulla di Figura 8.1 per visualizzare punti
;sullo schermo grafico ad alta risoluzione

          SCHERMO                ;inizializza lo schermo grafico 200 per 640
RIPETE:   MOV     SI,0           ;inizio della tabella
          MOV     AX,POS         ;trasferisce il valore dell'angolo in AL
          CMP     AX,180         ;angolo maggiore di 180 gradi?
          JLE     NEWQUAD        ;se è minore, l'angolo è nel 1° o nel 2° quadrante
          SUB     AX,180         ;angolo corretto se è maggiore o uguale a 180
NEWQUAD:  CMP     AX,90          ;angolo maggiore di 90 gradi?
          JLE     SECQUAD        ;se è maggiore, l'angolo è nel 2° quadrante
          NEG     AX             ;angolo complementato
          ADD     AX,180         ;angolo corretto se è maggiore o uguale a 90
```

```
SECQUAD:  ADD     SI,AX            ;calcolo del valore dell'indice
          SHL     SI,1             ;indice di tipo word (x2)
          MOV     AL,BYTE PTR SENO[SI] ;preleva il valore del seno
          CMP     POS,180          ;se il valore è maggiore di 180, somma lo spiazzamento
          JGE     SPIAZZ
          NEG     AL               ;altrimenti, complemeta il valore
          ADD     AL,100           ;somma 100 al valore, per un corretto
          JMP     OK               ;spiazzamento sullo schermo
SPIAZZ:   ADD     AL,99
OK:       MOV     BASSO,AL         ;memorizzazione del valore da visualizzare
          DISPLAY                  ;visualizzazione
          ADD     POS,1            ;prossimo angolo
          CMP     POS,360          ;si è arrivati a 360 gradi?
          JLE     RIPETE           ;se no, continua
;fine dell'esempio di visualizzazione di un'onda sinusoidale

;attendere la pressione di un tasto prima di ripristinare lo
;schermo in modalità normale
          MOV     AH,07            ;parametro per lettura tastiera
          INT     21H              ;lettura tastiera
          MOV     AH,00            ;parametro per lo schermo
          MOV     AL,03            ;modalità colore 25 per 80
          INT     10H              ;ripristino schermo

          RET                      ;il controllo ritorna al DOS
PROCEDURA ENDP                     ;fine della procedura
CODICE    ENDS                     ;fine del segmento di codice

          END                      ;fine del programma
```

La Figura 9.4 rappresenta l'immagine dell'onda sinusoidale come appare sullo schermo. Il coprocessore garantisce un calcolo e una visualizzazione rapida dei punti sullo schermo.

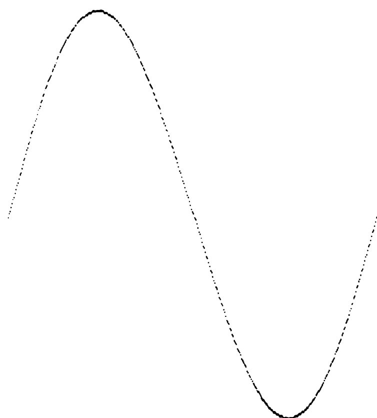


Figura 9.4 Diagramma di un'onda sinusoidale

Nel prossimo paragrafo vengono confrontate le velocità di visualizzazione di un grafico che sono ottenibili nel caso di programmazione in linguaggio assemblatore e in un linguaggio di alto livello.

9.4 Approssimazione di un'onda con lo sviluppo in serie di Fourier

Il seguente programma di grafica indica la velocità e la precisione che può essere raggiunta con l'80286/80386 e con i coprocessori Intel. Questo esempio è stato scritto ed eseguito su un PC AT IBM, dotato di microprocessore 80286 a 9 MHz e di un coprocessore 80287 che opera ad una velocità standard. Se viene utilizzato il microprocessore 80386, insieme al coprocessore 80387, si verifica un incremento di velocità di esecuzione di un fattore 10.
È stato osservato dal matematico Jean Baptiste Fourier (1768-1830) che quasi ogni forma d'onda periodica può essere approssimata sommando un certo numero di armoniche sinusoidali. L'equazione di Fourier viene espressa nel modo seguente:

$$y = A + A_1\sin(\omega t) + A_2\sin(2\omega t) + A_3\sin(3\omega t) + A_4\sin(4\omega t) + \ldots$$

Alcune forme d'onda vengono approssimate solo da armoniche di frequenza data da un multiplo pari o dispari della fondamentale, ma può anche capitare che siano presenti sia armoniche di ordine pari che dispari.
In questo esempio, approssimiamo un'onda quadra con uno sviluppo in serie di Fourier. Maggiore è il numero dei termini che costituiscono la serie, maggiore è la precisione del risultato finale. Per un'onda quadra, l'equazione generale della serie di Fourier è la seguente:

$$y = \sin(\omega t) + 1/3\,\sin(3\omega t) + 1/5\,\sin(5\omega t) + 1/7\,\sin(7\omega t) + \ldots$$

Dunque, solo le armoniche di ordine dispari contribuiscono al risultato finale. Si tenga presente che, se l'equazione contiene una sola armonica, il risultato è un'onda sinusoidale. Inoltre, si noti che ogni termine di ordine crescente viene moltiplicato per un coefficiente frazionario via via decrescente (in altre parole, ogni armonica di ordine superiore influenza sempre di meno la forma d'onda).
Ogni termine della serie di Fourier viene calcolato separatamente dal programma e poi sommato al risultato parziale. Quindi, se la serie si compone di 500 armoniche, vengono calcolati 500 valori distinti della funzione seno (rapportati alla risoluzione dello schermo) e vengono sommati insieme per determinare un unico punto da visualizzare sullo schermo. Questo procedimento deve essere ripetuto per ogni punto che deve essere visualizzato sullo

schermo. Quindi, nel nostro caso, quanto detto equivale a 500 elaborazioni per 360 punti, cioè 180 000 elaborazioni! Se, ad esempio, una calcolatrice impiega 5 minuti per calcolare 50 termini, sono necessari 50 minuti per calcolare 500 termini. Cinque minuti moltiplicato per 360 dà come risultato 18 000 minuti, cioè 12.5 giorni consecutivi. Più avanti verificheremo quanto tempo occorre per eseguire le precedenti elaborazioni in linguaggi di alto livello come BASIC, Pascal o APL.
Esaminiamo il programma che approssima con la serie di Fourier, e visualizza sullo schermo, un'onda quadra:

```
;per calcolatori con coprocessori matematici 80287/80387
;programma che genera un'onda quadra sommando insieme un certo
;numero di termini di una serie di Fourier. La forma d'onda
;risultante viene visualizzata su uno schermo ad alta risoluzione
;500 armoniche richiedono circa 1.8 minuti per essere calcolate

PAGE ,132                           ;dimensionamento pagina

 .8087                              ;direttiva per presenza coprocessore

ARMONICA  MACRO                     ;;numero decimale di 4 cifre in ingresso
          LOCAL   ANCORA,OK         ;;da tastiera
          PUSH    AX                ;;salvataggio dei registri
          PUSH    BX
          PUSH    CX
          PUSH    DX
          MOV     DX,0              ;;DX a zero
          MOV     CX,4              ;;quattro cifre al massimo in ingresso
ANCORA:   MOV     AH,1              ;;parametro di interruzione DOS
          INT     21H               ;;lettura caratteri ASCII da tastiera
          CMP     AL,30H            ;;limite inferiore per il numero in ingresso
          JL      OK                ;;lettura dato eseguita
          CMP     AL,39H            ;;limite superiore per il numero in ingresso
          JG      OK                ;;lettura dato eseguita
          AND     AX,000FH          ;;isola i quattro bit meno significativi (cifra esadecimale)
          PUSH    AX                ;;salva la cifra sullo stack
          MOV     AX,DX             ;;risultato parziale in AX
          MOV     BX,10             ;;moltiplicazione per 10
          MUL     BX
          MOV     DX,AX             ;;DX contiene il risultato parziale
          POP     AX
          ADD     DX,AX             ;;somma con l'ultima cifra inserita in ingresso
          LOOP    ANCORA
OK:       MOV     ARMON,DX          ;;numero di armoniche
          POP     DX                ;;ripristino registri
          POP     CX
          POP     BX
          POP     AX
          ENDM

SCHERMO   MACRO                     ;;inizializza lo schermo ad alta
          PUSH    AX                ;;risoluzione in b/n 200 per 640
          MOV     AH,00
          MOV     AL,06
          INT     10H
          POP     AX
          ENDM
```

```
DISPLAY   MACRO                   ;;macro per visualizzare i punti
          PUSH    AX
          MOV     AH,12
          MOV     AL,01
          MOV     CX,ANGOLO
          ADD     CX,140          ;;immagine al centro dello schermo
          MOV     DH,00
          INT     10H
          POP     AX
          ENDM

STACK     SEGMENT PARA STACK
          DB      48 DUP ('STACK ')
STACK     ENDS

DATI      SEGMENT PARA 'DATI'
ANGOLO    DW      0               ;valore originale dell'angolo (da 0 a 360)
QUATTRO   DW      4               ;una costante
MOLTIP    DW      50              ;converte il seno da 0 a 100
RADIANTE  DW      180             ;una costante
RIDOTTO   DW      360             ;una costante
TEMP1     DW      ?
TEMP2     DW      ?
TEMP3     DW      ?
PARSTATO  DW      ?               ;parola di stato dell'80287
SENO      DD      361 DUP (0)     ;memoria per i risultati reali
SENOINT   DW      361 DUP (0)     ;memoria per i risultati interi
MESS1     DB      'INSERIRE IL NUMERO DI ARMONICHE DA SOMMARE, (da 0 a 9999): $',13H
MESS2     DB      'ELABORAZIONE IN CORSO $'
ARMON     DW      ?               ;numero di armoniche
DATI      ENDS

CODICE    SEGMENT PARA 'CODICE'   ;definisce il segmento di codice
PROCEDURA PROC    FAR           , ;inizio della procedura
          ASSUME  CS:CODICE,DS:DATI,SS:STACK,ES:DATI
          PUSH    DS              ;salva DS sullo stack
          SUB     AX,AX           ;azzera AX
          PUSH    AX              ;salva 0 sullo stack
          MOV     AX,DATI         ;indirizzo di DATI in AX
          MOV     DS,AX           ;indirizzo di DATI in DS

          SCHERMO                 ;cancella lo schermo
          LEA     DX,MESS1        ;visualizza il messaggio di richiesta dati
          MOV     AH,9
          INT     21H
          ARMONICA                ;accettazione dati in ingresso
          SCHERMO                 ;cancella lo schermo e aspetta
          LEA     DX,MESS2        ;visualizza il secondo messaggio
          MOV     AH,9
          INT     21H

          MOV     SI,0            ;SI è l'indice di SENO
NEXT:     MOV     TEMP1,01H       ;analisi angolo
ANCORA:   MOV     AX,ANGOLO       ;angolo corrente in AX
          MOV     DX,TEMP1        ;armonica corrente di Fourier
          SHL     DX,1            ;moltiplicata per 2
          SUB     DX,1            ;sottrazione di 1
          MOV     TEMP2,DX        ;memorizzazione del fattore
          MUL     TEMP2           ;angolo moltiplicato per il fattore
          DIV     RIDOTTO         ;divisione dell'angolo per 360
          MOV     TEMP3,DX        ;memorizzazione gradi restanti
```

```
        FINIT                  ;inizializza il coprocessore
        FILD    RADIANTE       ;inizic conversione da gradi a radianti
        FLDPI                  ;π nello stack
        FDIV    ST(0),ST(1)    ;divisione per avere .0174.....+
        FILD    TEMP3          ;angolo nello stack (in gradi)
        FMUL                   ;angolo in radianti

COMMENT / Se si usa il coprocessore 80387, che supporta le
funzioni trigonometriche estese, la seguente sezione di
codice - delimitata dagli asterischi - può essere
sostituita con l'istruzione FSIN  /

;*****************************************************************************
        FLDPI                  ;π in cima allo stack                          *
        FIDIV   QUATTRO        ;divide per 4, risultato in cima allo stack    *
        FXCH                   ;scambia la posizione dell'angolo              *
        FPREM                  ;riduzione da 0 a π/4 per calcolo tangente     *
        FSTSW   PARSTATO       ;parola di stato corrente in PARSTATO          *
        FWAIT                  ;sincronizzazione                              *
        MOV     AX,PARSTATO    ;parola di stato in AX                         *
        TEST    AH,00000010B   ;test sui bit: se zero, angolo<46 gradi        *
        JZ      CALTAN         ;calcolo tangente                              *
        FSUBP   ST(1),ST(0)    ;sottrazione da 45 gradi                       *
CALTAN:                                                                       *
        FPTAN                  ;tangente dell'angolo                          *
        TEST    AH,01000010B   ;± sin (π/4 - ip)                              *
        JPE     TST2           ;se si, altro test                             *
        JMP     FIXIT          ;altrimenti, scambio ST e ST1                  *
TST2:   TEST    AH,00000000B   ;± sin (ip)                                    *
        JPE     CALSENO        ;se entrambi sono zero o uno                   *
FIXIT:  FXCH                   ;scambio, utilizza funzione coseno             *
CALSENO:                       ;altrimenti, usa la funzione seno              *
        FMUL    ST(0),ST       ;funzione trig. per estrarre                   *
        FXCH    ST(1)          ;dalla tangente il valore corretto             *
        FLD     ST(0)          ;del seno                                      *
        FMUL    ST(0),ST                                                      *
        FADD    ST(0),ST(2)                                                   *
        FSQRT                                                                 *
        FDIVP   ST(1),ST                                                      *
        TEST    AH,00000001B   ;uno zero significa un risultato positivo      *
        JZ      SEGNOPOS                                                      *
        FCHS                   ;altrimenti, cambio segno del risultato        *
;*****************************************************************************

SEGNOPOS: FIMUL MOLTIP         ;moltiplica il risultato per il fattore di schermo
        FIDIV   TEMP2          ;divisione per il fattore di armonica
        FADD    SENO[SI]       ;somma il risultato
        FSTP    SENO[SI]       ;salva in memoria la nuova somma reale
        FWAIT                  ;sincronizzazione

        INC     TEMP1          ;preparativi per la prossima armonica
        MOV     CX,ARMON       ;numero massimo di armoniche in CX
        CMP     TEMP1,CX       ;confronta con il valore massimo
        JG      ANG            ;se maggiore, fine e analisi altro angolo
        JMP     ANCORA         ;altrimenti, continua l'accumulo di armoniche

ANG:    ADD     SI,04H         ;punta alla successiva locazione di SENO
        INC     ANGOLO         ;angolo successivo
        CMP     ANGOLO,360     ;confronto con 360 gradi
        JG      TRASF          ;se maggiore, trasferimento del dato
        JMP     NEXT           ;se non maggiore, ripete
```

```
;copia 361 numeri reali da SENO a SENOINT convertendoli in valori
;interi compresi tra ± 100.
TRASF:    MOV    SI,00           ;registri indice a 0
          MOV    DI,00
          LEA    BX,SENO         ;carica l'indirizzo di due tabelle
          LEA    BP,SENOINT
          MOV    CX,361          ;inizio trasferimento di 361 valori
NOSTOP:   FLD    SENO[SI]        ;carica il numero reale sullo stack
          FISTP  SENOINT[SI]     ;estrazione e memorizzazione del numero intero
          ADD    SI,04           ;aggiornamento indice alla tabella di reali
          ADD    DI,02           ;aggiornamento indice alla tabella di interi
          LOOP   NOSTOP

;visualizzazione dei valori interi sullo schermo
PLOT:     SCHERMO                ;inizializza lo schermo grafico 200 per 640
          LEA    BP,SENOINT      ;indirizzo di SENOINT in BP
          MOV    SI,0            ;inizio tabella
          MOV    ANGOLO,0        ;angolo da visualizzare
RIPETE:   MOV    DL,BYTE PTR SENOINT[SI] ;valore da visualizzare in DL
          NEG    DL              ;sottrazione del valore da 100
          ADD    DL,100          ;coordinata verticale
          DISPLAY                ;visualizzazione
          ADD    SI,02           ;prossimo locazione di memoria
          ADD    ANGOLO,1        ;prossimo angolo
          CMP    ANGOLO,360      ;si è arrivati a 360 gradi?
          JLE    RIPETE          ;se no, continuazione

;attesa di un carattere in ingresso prima di ripristinare lo
;stato dello schermo
          MOV    AH,07           ;parametro di tastiera
          INT    21H             ;lettura da tastiera
          MOV    AH,00           ;parametro di schermo
          MOV    AL,03           ;modalità colore 25 per 80
          INT    10H             ;ripristino dello stato dello schermo

          RET                    ;il controllo ritorna al DOS
PROCEDURA ENDP                   ;fine della procedura
CODICE    ENDS                   ;fine del segmento di codice

          END                    ;fine del programma
```

Questo programma può sembrare molto complicato. In realtà, si compone di alcune parti di programmi che abbiamo già discusso precedentemente. Il programma utilizza tre macro: ARMONICA, SCHERMO e DISPLAY. Queste ultime due non hanno bisogno di essere analizzate in dettaglio (in quanto corrispondono alle stesse macro utilizzate nel programma che abbiamo presentato nel paragrafo precedente): inizializzano lo schermo grafico e visualizzano un punto sullo schermo. Al contrario, la macro ARMONICA è molto interessante. Per rendere il programma il più flessibile possibile, l'utente deve poter inserire da tastiera il numero di armoniche della serie di Fourier. Ci sono molti modi per raggiungere questo scopo, ma la tecnica che abbiamo scelto in questo programma è semplice e facile da capire.
Riportiamo qui di seguito il nucleo della macro, che permette di inserire da tastiera un numero decimale di quattro cifre. Questo numero viene accumulato nel registro DX.

```
         MOV    DX,0        ;;DX a zero
         MOV    CX,4        ;;quattro cifre al massimo in ingresso
ANCORA:  MOV    AH,1        ;;parametro di interruzione DOS
         INT    21H         ;;lettura caratteri ASCII da tastiera
         CMP    AL,30H      ;;limite inferiore per il numero in ingresso
         JL     OK          ;;lettura dato eseguita
         CMP    AL,39H      ;;limite superiore per il numero in ingresso
         JG     OK          ;;lettura dato eseguita
         AND    AX,000FH    ;;isola i quattro bit meno significativi (cifra
                            ;;esadecimale)
         PUSH   AX          ;;salva la cifra sullo stack
         MOV    AX,DX       ;;risultato parziale in AX
         MOV    BX,10       ;;moltiplicazione per 10
         MUL    BX
         MOV    DX,AX       ;;DX contiene il risultato parziale
         POP    AX
         ADD    DX,AX       ;;somma con l'ultima cifra inserita in ingresso
         LOOP   ANCORA
OK:      MOV    ARMON,DX    ;;numero di armoniche
```

Il registro DX viene inizializzato a zero per garantire una corretta acquisizione del dato in ingresso, mentre il registro CX memorizza il contatore di ciclo.
Ad ogni esecuzione del ciclo, è possibile accumulare in DX una nuova cifra, ma il valore 4 in CX limita a quattro il numero di cifre. Il valore 1 nel registro AH permette la lettura da tastiera, quando viene invocata l'interruzione INT 21H. Naturalmente, il dato in ingresso è nel formato ASCII. La macro si assicura che si tratti di un numero decimale (valore ASCII compreso tra 30H e 39H), altrimenti interrompe l'operazione di acquisizione dati. L'operazione di prodotto logico (AND) tra il contenuto di AX e 000F permette di salvare solo la parte decimale del numero ASCII, per cui AX contiene un numero compreso tra 0 e 9. Il valore di AX viene poi salvato sullo stack e il valore di DX viene memorizzato nel registro AX. Il valore 10 posto in BX rappresenta un moltiplicatore. Qualunque numero sia contenuto in DX e poi sia stato trasferito in AX viene ora moltiplicato per 10. Il risultato della moltiplicazione viene memorizzato in AX e trasferito di nuovo in DX. Il contenuto del registro AX viene estratto dallo stack e sommato a DX, in modo da ottenere il numero decimale corretto. Se è stata inserita da tastiera una cifra sbagliata, oppure se sono state accumulate quattro cifre, la macro termina.
Consideriamo il seguente esempio. Se era stata inserita da tastiera la cifra 9, il registro DX contiene il valore 9. Supponiamo che il programma esegua una seconda volta il ciclo e che venga inserita da tastiera la cifra 5. Quindi il valore ASCII complessivo è 35, che rappresenta una cifra corretta. Su questo valore viene eseguita l'operazione AND con 000F e il risultato viene memorizzato sullo stack. Il contenuto di DX (cioè 9) viene memorizzato nel

registro AX e viene moltiplicato per 10, ottenendo il valore 90, che viene trasferito nel registro DX. Il contenuto di AX viene estratto dallo stack e viene sommato a DX, per cui il registro DX contiene il valore 95. Se viene battuto il tasto di ritorno a capo, il valore 95 viene memorizzato nella variabile ARMON (armonica) e la macro termina.
Prima di esaminare il programma, analizziamo il contenuto del segmento dati:

```
ANGOLO     DW   0              ;valore originale dell'angolo (da 0 a 360)
QUATTRO    DW   4              ;una costante
MOLTIP     DW   50             ;converte il seno da 0 a 100
RADIANTE   DW   180            ;una costante
RIDOTTO    DW   360            ;una costante
TEMP1      DW   ?
TEMP2      DW   ?
TEMP3      DW   ?
PARSTATO   DW   ?              ;parola di stato dell'80287
SENO       DD   361 DUP (0)    ;memoria per i risultati reali
SENOINT    DW   361 DUP (0)    ;memoria per i risultati interi
MESS1      DB   'INSERIRE IL NUMERO DI ARMONICHE DA SOMMARE,
           DB   (da 0 a 9999): $',13H
MESS2      DB   'ELABORAZIONE IN CORSO $'
ARMON      DW   ?              ;numero di armoniche
```

ANGOLO contiene la posizione orizzontale dello schermo in cui il punto deve essere visualizzato. Come nel precedente esempio, questo programma visualizza 361 punti orizzontali (pixel). QUATTRO, MOLTIP, RADIANTE e RIDOTTO sono valori costanti, mentre TEMP1, TEMP2 e TEMP3 costituiscono tre locazioni di memoria destinate a memorizzare risultati di elaborazioni intermedie. PARSTATO memorizza la parola di stato del coprocessore e aiuta a stabilire in quale dei quattro quadranti l'angolo si trovi. SENO è una tabella di 361 somme reali (ogni somma costituisce il valore della serie di Fourier), che vengono poi convertite in numeri interi, memorizzate in un'altra tabella (SENOINT) e rappresentano le posizioni verticali sullo schermo dei punti dell'onda. MESS1 segnala all'utente che può inserire da tastiera il numero di armoniche, mentre MESS2 informa che l'elaborazione è in corso di svolgimento. Non viene visualizzato nulla sullo schermo, fino a quando tutti i punti non sono stati calcolati. Nel caso di onde con 500 armoniche, il tempo impiegato per eseguire le elaborazioni è di 1.8 minuti su una macchina a 9 MHz.
La prima parte del codice richiede all'utente di inserire i dati da tastiera:

```
SCHERMO                  ;cancella lo schermo
LEA        DX,MESS1      ;visualizza il messaggio di richiesta dati
MOV        AH,9
INT        21H
ARMONICA                 ;accettazione dati in ingresso
```

```
SCHERMO                     ;cancella lo schermo e aspetta
LEA        DX,MESS2         ;visualizza il secondo messaggio
MOV        AH,9
INT        21H
```

La macro SCHERMO inizializza lo schermo alla modalità grafica ad alta risoluzione e lo cancella. Viene visualizzato il messaggio MESS1, che richiede all'utente di inserire da tastiera il numero di armoniche, che viene memorizzato nella variabile ARMON. È possibile specificare da 0 a 9999 armoniche (comunque, come si può notare dalle prossime figure, un numero di armoniche pari a 500 permette già di produrre un'onda quadra di elevata precisione). Il messaggio MESS2 informa l'utente che gli ingressi sono stati accettati e che è in corso l'elaborazione dei risultati.
Questo programma si differenzia dal precedente, in cui il calcolo delle coordinate è stato effettuato solo per il primo quadrante, data la simmetria della forma d'onda nei restanti tre quadranti. In questo esempio (onda non simmetrica), invece, ognuno dei 361 punti viene calcolato, memorizzato e visualizzato. Questo significa che tutti gli angoli devono essere ridotti ad un valore compreso tra 0 e 360. Un'altra considerazione interessante è che aumentando il numero di armoniche, cresce il moltiplicatore dell'angolo.
Analizziamo la seguente parte del programma:

```
           MOV   SI,0          ;SI è l'indice di SENO
NEXT:      MOV   TEMP1,01H     ;analisi angolo
ANCORA:    MOV   AX,ANGOLO     ;angolo corrente in AX
           MOV   DX,TEMP1      ;armonica corrente di Fourier
           SHL   DX,1          ;moltiplicata per 2
           SUB   DX,1          ;sottrazione di 1
           MOV   TEMP2,DX      ;memorizzazione del fattore
           MUL   TEMP2         ;angolo moltiplicato per il fattore
           DIV   RIDOTTO       ;divisione dell'angolo per 360
           MOV   TEMP3,DX      ;memorizzazione gradi restanti
```

Queste istruzioni definiscono le azioni che vengono compiute dal programma. SI viene inizializzato a 0 e rappresenta l'indice che referenzia la tabella SENO. TEMP1 contiene inizialmente il valore 1 e referenzia l'armonica corrente di Fourier (varia da 1 ad ARMON per ogni ANGOLO e assume solo valori dispari nel caso di onda quadra), mentre ANGOLO memorizza la posizione orizzontale corrente (varia tra 0 e 360 gradi). TEMP1 viene trasferito in DX, moltiplicato per 2 (traslazione a sinistra) e poi ridotto di una unità. DX ora contiene un numero dispari, che viene salvato in TEMP2 perché sia utilizzabile successivamente. Il contenuto di ANGOLO viene trasferito in AX e moltiplicato per il precedente fattore. Il prodotto a 32 bit, contenuto nella coppia di registri AX/DX, viene diviso per RIDOTTO (360), in modo da ridurre l'angolo ad un valore compreso tra 0 e 360 gradi. Solo il resto della divisione viene salvato nel registro DX. I valori interi contenuti nel registro

AX non sono di alcuna utilità, in quanto rappresentano multipli di 360 gradi (si tratta di una forma d'onda periodica, che si ripete ogni 360 gradi). Il resto viene memorizzato nella variabile TEMP3.
A questo punto, TEMP3 contiene un angolo compreso tra 0 e 360 gradi. Le seguenti istruzioni convertono l'angolo da gradi a radianti:

```
FINIT                      ;inizializza il coprocessore
FILD     RADIANTE          ;inizio conversione da gradi a radianti
FLDPI                      ;π nello stack
FDIV     ST(0),ST(1)       ;divisione per avere .0174.....+
FILD     TEMP3             ;angolo nello stack (in gradi)
FMUL                       ;angolo in radianti
```

Al termine della conversione, l'angolo viene salvato in cima allo stack del coprocessore. Le istruzioni comprese tra gli asterischi permettono di calcolare il seno di qualunque angolo compreso tra 0 e 2π radianti (tra 0 e 360 gradi). Se si usa il coprocessore 80387, tutte queste istruzioni vengono sostituite con la sola istruzione FSIN.
Si tenga presente che tutti gli angoli sono stati precedentemente ridotti ad un valore compreso tra 0 e 360 gradi. Se si vuole calcolare correttamente il valore della funzione tangente, è necessario ridurre ogni angolo, che si trova in cima allo stack, da un valore compreso tra 0 e 2π ad un valore compreso tra 0 e $\pi/4$ (0 e 45 gradi). In un precedente esempio, abbiamo visto come sia possibile calcolare il seno di un angolo – compreso tra 0 e 90 gradi – utilizzando semplici funzioni trigonometriche. In questo programma, invece, abbiamo utilizzato un approccio leggermente differente.

PAROLA DI STATO

Il registro a 16 bit della parola di stato (status word) dell'80287 descrive lo stato corrente del coprocessore. Elenchiamo qui di seguito le condizioni di eccezione per il coprocessore:

bit 0	IE	Operazione non valida
bit 1	DE	Operazione non normalizzata
bit 2	ZE	Divisione per zero
bit 3	OE	Overflow
bit 4	UE	Underflow
bit 5	PE	Precisione
bit 6		(riservato)
bit 7	IR	Richiesta di interruzione
bit 8	C0	Bit di condizione
bit 9	C1	Bit di condizione
bit 10	C2	Bit di condizione
bit 11	ST	Bit puntatore cima dello stack

bit 12	ST	Bit puntatore cima dello stack
bit 13	ST	Bit puntatore cima dello stack
bit 14	C3	Bit di condizione
bit 15	B	Occupato

I bit di condizione della parola di stato permettono di assumere alcune decisioni nel corso del programma.
Esaminiamo un'altra parte di programma:

```
FLDPI                           ;π in cima allo stack
FIDIV     QUATTRO               ;divide per 4, risultato in cima allo stack
FXCH                            ;scambia la posizione dell'angolo
FPREM                           ;riduzione da 0 a π/4 per calcolo tangente
FSTSW     PARSTATO              ;parola di stato corrente in PARSTATO
FWAIT                           ;sincronizzazione
MOV       AX,PARSTATO           ;parola di stato in AX
TEST      AH,00000010B          ;test sui bit: se zero, angolo < 46 gradi
JZ        CALTAN                ;calcolo tangente
FSUBP     ST(1),ST(0)           ;sottrazione da 45 gradi
```

La costante numerica π viene caricata in cima allo stack e il valore dell'angolo, precedentemente memorizzato in cima allo stack, scende di una posizione. π viene divisa per la costante QUATTRO e il risultato viene memorizzato in cima allo stack. A questo punto, in cima allo stack c'è il valore $\pi/4$, mentre ST(1) contiene il valore dell'angolo, che è stato calcolato in una precedente operazione. ST e ST(1) vengono scambiati e viene eseguita l'istruzione FPREM, che divide ST per ST(1). Questo significa che l'angolo, che ora si trova in cima allo stack, viene diviso per $\pi/4$. Supponiamo ad esempio che l'angolo sia 4.88692 radianti. Dividendo questo numero per $\pi/4$, si ottiene 6.22222. Il resto vale circa .22222, cioè circa 12.73 gradi, che corrisponde ad un angolo reale di 283.73 gradi ($\sin(283.73) = -\sin(90-12.73)$).

Tabella 9.4 Risultati sperimentali per il calcolo degli ottanti nel programma di calcolo del seno

Ottante	Angolo	Angolo/(π/4)	Quoziente	Resto
1	0.1745	0.22222	0	.22222
2	1.3963	1.77778	1	.77778
3	1.7453	2.22222	2	.22222
4	2.9671	3.77778	3	.77778
5	3.3161	4.22222	4	.22222
6	4.5379	5.77778	5	.77778
7	4.8869	6.22222	6	.22222
8	6.1087	7.77778	7	.77778

Per sapere che il segno dell'angolo è negativo e che l'angolo stesso deve essere sottratto a 90 gradi ($\pi/2$), si procede nel modo seguente. Utilizzando un fattore di scala di $\pi/4$ per un angolo compreso tra 0 e 2π (tra 0 e 360 gradi), viene eseguita una divisione dell'angolo in ottanti. Esaminando la Tabella 9.4 si può notare che esiste una relazione lineare tra l'ottante e il quoziente. Dunque, si può utilizzare il quoziente per determinare in quale ottante si trovi l'angolo e, una volta noto l'ottante, è possibile stabilire quale funzione trigonometrica applicare.
A questo punto, è possibile utilizzare l'istruzione FDIV invece di FPREM per eseguire la divisione. Così facendo, il valore intero corrispondente al quoziente indica l'ottante e il resto viene utilizzato dalla funzione tangente per calcolare il seno dell'angolo. Comunque, la stessa informazione è memorizzata nella parola di stato dopo l'esecuzione dell'istruzione FPREM, come viene indicato nella Tabella 9.5.
Viene eseguito un primo test su C1. Il contenuto della variabile PARSTATO viene trasferito nel registro AX e viene effettuato un confronto tra il numero binario 00000010B (2H) e il contenuto di AH. Se il bit C1 vale 1, la verifica è positiva e non viene eseguito alcun salto. Il resto – cioè ST(0) – viene sottratto a $\pi/4$ – cioè a ST(1) – e viene eseguita un'operazione di estrazione del valore contenuto in cima allo stack. In questo modo, il nuovo angolo si trova in cima allo stack.

```
CALTAN:
            FPTAN                       ;tangente dell'angolo
            TEST     AH,01000010B       ;± sin (π/4 - ip)
            JPE      TST2               ;se sì, altro test
            JMP      FIXIT              ;altrimenti, scambio ST e ST1
TST2:       TEST     AH,00000000B       ;± sin (ip)
            JPE      CALSENO            ;se entrambi sono zero o uno
FIXIT:      FXCH                        ;scambio, utilizza funzione coseno
```

Tabella 9.5 Test sull'ottante per il programma di calcolo del seno

Ottante	Quoziente	C0	C3	C1	Funzione
1	0	0	0	0	sin(*angolo*)
2	1	0	0	1	cos($\pi/4$ – *angolo*)
3	2	0	1	0	cos(*angolo*)
4	3	0	1	1	sin($\pi/4$ – *angolo*)
5	4	1	0	0	– sin(*angolo*)
6	5	1	0	1	– cos($\pi/4$ – *angolo*)
7	6	1	1	0	– cos(*angolo*)
8	7	1	1	1	– sin($\pi/4$ – *angolo*)

Viene calcolata la tangente dell'angolo. Vengono eseguiti due differenti test: se C3 e C1 valgono entrambi 0 oppure 1, viene eseguita la funzione trigonometrica per il calcolo del seno; altrimenti, viene eseguita la funzione coseno, scambiando ST con ST(1). Questo procedimento è simile a quello che ha permesso di visualizzare un'onda sinusoidale.

```
CALSENO:                                 ;altrimenti, usa la funzione seno
          FMUL     ST(0),ST              ;funzione trig. per estrarre
          FXCH     ST(1)                 ;dalla tangente il valore corretto
          FLD      ST(0)                 ;del seno
          FMUL     ST(0),ST
          FADD     ST(0),ST(2)
          FSQRT
          FDIVP    ST(1),ST
```

Dopo l'esecuzione dell'istruzione FDIVP, in cima allo stack si trova il valore corretto del seno dell'angolo, a meno del segno. Il segno dell'angolo può essere determinato esaminando il bit C0. Se C0 vale 1, il segno è negativo, mentre se vale 0 il segno è positivo.

```
          TEST     AH,00000001B          ;uno zero significa un risultato positivo
          JZ       SEGNOPOS
          FCHS                           ;altrimenti, cambio segno del risultato
```

Il risultato che si trova in cima allo stack deve essere rapportato in scala e sommato ai risultati relativi alle precedenti armoniche:

```
SEGNOPOS:FIMUL     MOLTIP                ;moltiplica il risultato per il fattore di
                                         ;schermo
          FIDIV    TEMP2                 ;divisione per il fattore di armonica
          FADD     SENO[SI]              ;somma il risultato
          FSTP     SENO[SI]              ;salva in memoria la nuova somma
                                         ;reale
          FWAIT                          ;sincronizzazione
```

Si tenga presente che TEMP2 contiene il fattore moltiplicativo di armonica, memorizzato all'inizio del programma. Nel caso di onde quadre, questo fattore viene utilizzato come moltiplicatore per l'angolo e come divisore dell'ampiezza di ogni armonica. L'istruzione FADD somma il totale corrente al valore contenuto in cima allo stack e memorizza il nuovo risultato nella stessa locazione di memoria.
Se consideriamo 50 armoniche, è necessario sommare insieme 50 valori distinti, prima che SI venga incrementato alla posizione orizzontale successiva. Il numero di iterazioni viene controllato dalla variabile TEMP1.

```
     INC      TEMP1           ;preparativi per la prossima armonica
     MOV      CX,ARMON        ;numero massimo di armoniche in CX
```

```
        CMP     TEMP1,CX        ;confronta con il valore massimo
        JG      ANG             ;se maggiore, fine e analisi altro angolo
        JMP     ANCORA          ;altrimenti, continua l'accumulo di armoniche
```

ARMON contiene il numero complessivo di armoniche.
Viene eseguita una semplice verifica per stabilire se devono essere calcolate altre armoniche. Se la verifica dà esito positivo, viene effettuato un salto all'etichetta ANCORA, altrimenti il programma è pronto a considerare la successiva posizione orizzontale e a iniziare di nuovo le elaborazioni.

```
ANG:    ADD     SI,04H          ;punta alla successiva locazione di SENO
        INC     ANGOLO          ;angolo successivo
        CMP     ANGOLO,360      ;confronto con 360 gradi
        JG      TRASF           ;se maggiore, trasferimento del dato
        JMP     NEXT            ;se non maggiore, ripete
```

Ogni elemento della tabella SENO è stato definito di tipo doubleword (DD), per cui SI viene incrementato di 4 byte per puntare correttamente alla successiva locazione di memoria. ANGOLO contiene la posizione orizzontale, cioè un valore compreso tra 0 e 360 gradi. Una volta che tutte le armoniche sono state calcolate per una certa posizione, il valore di ANGOLO viene incrementato di una unità e il processo continua fino a quando non si sono esaurite tutte le 361 posizioni.
Poiché i valori accumulati in SENO devono essere più precisi possibile, vengono memorizzati come numeri reali. Prima che i risultati siano visualizzati sullo schermo, questi numeri vengono convertiti in valori interi:

```
TRASF:  MOV     SI,00           ;registri indice a 0
        MOV     DI,00
        LEA     BX,SENO         ;carica l'indirizzo di due tabelle
        LEA     BP,SENOINT
        MOV     CX,361          ;inizio trasferimento di 361 valori
NOSTOP: FLD     SENO[SI]        ;carica il numero reale sullo stack
        FISTP   SENOINT[SI]     ;estrazione e memorizzazione del numero
                                ;intero
        ADD     SI,04           ;aggiornamento indice alla tabella di reali
        ADD     DI,02           ;aggiornamento indice alla tabella di interi
        LOOP    NOSTOP
```

I risultati finali contenuti in SENO (numeri reali) vengono manipolati ancora dal coprocessore, uno alla volta; vengono estratti dallo stack e memorizzati, nel formato intero, nella tabella SENOINT, i cui elementi sono stati definiti di tipo word (DW).
I valori contenuti in SENOINT vengono rapportati in scala e poi visualizzati sullo schermo. Il codice restante del programma esegue queste operazioni e attende che l'utente prema un tasto prima di cancellare lo schermo.
La Figura 9.5 mostra come appare l'immagine sullo schermo, quando il pro-

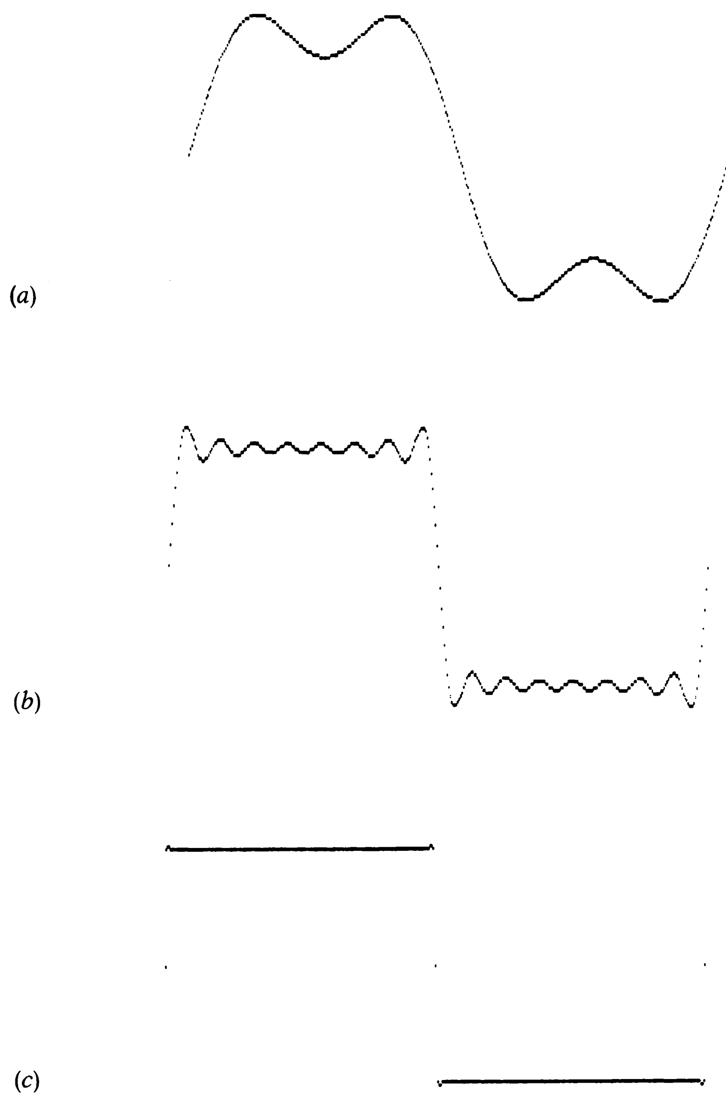


Figura 9.5 Immagini sullo schermo prodotte dal programma che utilizza la serie di Fourier: (*a*) 2 armoniche, (*b*) 8 armoniche, (*c*) 500 armoniche

gramma viene eseguito utilizzando tre diversi dati in ingresso, ovvero 2, 8 e 500 armoniche. L'esecuzione del programma è molto rapida, anche nel terzo caso.
Si è voluto verificare anche il tempo di esecuzione impiegato da programmi simili a quello che abbiamo descritto, ma scritti in BASIC, Pascal e APL. Il risultato di queste verifiche (valori medi) è stato riassunto in Tabella 9.6: si può quindi affermare che è stato decisamente conveniente scrivere il programma in linguaggio assemblatore.

Tabella 9.6 Tempi di esecuzione di differenti programmi di calcolo di onde quadre con serie di Fourier a 500 armoniche, su di un PC AT IBM con clock a 9 MHz

Linguaggio	**Tempo**
BASICA (non compilato)	69.75 minuti
APL (non compilato)	19.58 minuti
BASIC (compilato)	10.66 minuti
Pascal (compilato)	3.21 minuti
Assemblatore	1.80 minuti

10

Interfaccia con i linguaggi di alto livello

Molti programmatori fanno uso del linguaggio assemblatore per realizzare le funzioni che non sono supportate dai linguaggi di alto livello. Nel caso, ad esempio, di APL, BASIC, C, FORTRAN e Pascal esistono alcune limitazioni dovute al tipo di compilatore di cui è dotato il sistema. Il programmatore in un linguaggio di alto livello può quindi disporre solo delle utilità che sono state definite da altri programmatori o delle funzioni che vengono supportate dal compilatore.

Il primo compilatore Pascal IBM non supportava le funzioni grafiche, per cui l'unica soluzione possibile era quella di dotare il compilatore di un'estensione in linguaggio assemblatore (situazioni come questa sono molto comuni). Che si tratti di grafica, di joystick, di porte parallele, di porte seriali o di qualche altro supporto di interfaccia non disponibile, si ricorre sempre alla soluzione del linguaggio assemblatore.

Interfacciare un linguaggio di alto livello con il linguaggio assemblatore pone alcuni problemi, tra cui la risoluzione dei riferimenti esterni e il passaggio delle variabili dal programma chiamante al programma chiamato e viceversa. Nei prossimi esempi, verranno discusse dettagliatamente diverse soluzioni per realizzare le estensioni che interessano.

Esaminiamo, in particolare, come codificare un programma in linguaggio assemblatore che si interfacci con il joystick di un calcolatore IBM AT. Questo programma riceve dal programma principale un valore che specifica se campionare i pulsanti o i potenziometri del joystick. I risultati del campionamento vengono restituiti al programma chiamante scritto in un linguaggio di alto livello. Questo esempio indica, in particolare, come realizzare il passaggio dei parametri.

Una parte del codice di questo programma è comune a tutti i programmi di alto livello:

```
;codice per il campionamento di quattro pulsanti
                MOV   AH,84H          ;interfaccia joystick
                MOV   DX,0            ;DX=0 per9 informazione pulsanti
                INT   15H
                MOV   CL,04           ;rotazione di quattro bit
                ROR   AL,CL           ;bit 7-4 nei bit 3-0
                AND   AX,0FH          ;isola solo i 4 bit meno significativi
                JMP   SEND            ;ritorno dell'informazione

;codice per il campionamento di quattro potenziometri (A(X), A(Y), B(X) e B(Y))
POT:            MOV   AH,84H          ;interfaccia leva comando
                MOV   DX,01H          ;lettura potenziometri
                INT   15H
```

L'interfaccia con la leva di comando sul calcolatore IBM AT viene realizzata codificando la chiamata a interruzione INT 15H. AH viene inizializzato a 84H. Se DX contiene il valore 0, i dati che riguardano lo stato dei quattro pulsanti vengono restituiti nel registro AL. Se DX contiene il valore 1, vengono restituiti nei registri AX, BX, CX e DX i valori dei quattro potenziometri di gioco (due per ogni leva di comando). In questa parte del programma, l'informazione presente in DX è stata già passata con una precedente operazione. Viene presa una decisione, in base al valore contenuto in DX: se DX contiene il valore 0, l'informazione che riguarda il pulsante viene restituita nei quattro bit più significativi (7-4) del registro AL. Il programma esegue la rotazione di questa informazione nei quattro bit meno significativi (3-0) del registro AL e poi termina (viene eseguito un salto a SEND). Se DX contiene il valore 1, viene effettuato, invece, un salto all'etichetta POT. Quando viene invocata l'interruzione, i valori del potenziometro (valori interi compresi tra 0 e 270, in base alla particolare leva di comando utilizzata) vengono restituiti nei quattro registri generali.
Vengono ora presentati alcuni esempi di interfacciamento con i linguaggi di alto livello APL della STSC, Turbo Pascal della Borland, BASIC e C della Microsoft e FORTRAN e Pascal della IBM. I programmi di supporto scritti in linguaggio assemblatore per questi linguaggi di alto livello contengono molte similitudini, ma anche alcune differenze, per cui verranno presentati sempre in versione completa.

10.1 L'APL della STSC

Una delle migliori versioni del linguaggio APL per la famiglia di calcolatori IBM viene prodotta dalla STSC di Rockville, Maryland. Questa versione di APL offre un editing di video, una grafica completa, routine per la gestione

```
;per 80286/80386
;programma che richiede valori di pulsanti o di  potenziometri
;dalle leve di comando A & B; con interruzioni BIOS restituisce i
;valori ad un programma principale scritto in APL STSC

        PUBLIC  GIOCO
CODICE  SEGMENT 'CODICE'
        ASSUME  CS:CODICE
GIOCO   PROC    FAR             ;la procedura si chiama GIOCO
        PUSH    BP              ;salva BP sullo stack
        MOV     BP,SP           ;trasferisce lo stack pointer al registro BP

        MOV     SI,[BP]+22      ;informazione sull'operazione da programma principale
        MOV     DX,[SI]         ;salva l'informazione nel registro DX
        CMP     DX,0            ;se è zero, continua e legge i pulsanti
        JNE     POT             ;se non è zero,  salta e legge i potenziometri

;codice per il campionamento dei quattro pulsanti
        MOV     AH,84H          ;interfaccia leva di comando
        MOV     DX,0            ;DX=0 per informazione pulsanti
        INT     15H
        MOV     CL,04           ;rotazione di quattro bit
        ROR     AL,CL           ;bit 7-4 nei bit 3-0
        AND     AX,0FH          ;isola i quattro bit meno significativi
        JMP     SEND            ;restituzione dell'informazione

;codice per il campionamento di quattro potenziometri (A(X), A(Y), B(X) e B(Y))
POT:    MOV     AH,84H          ;interfaccia leva comando
        MOV     DX,01H          ;lettura potenziometri
        INT     15H

;codice per restituire i valori dei registri AX, BX, CX e DX.
SEND:   MOV     DI,[BP]+18      ;invio valore del potenziometro A(X)
        MOV     [DI],AX
        MOV     DI,[BP]+14      ;invio valore del potenziometro A(X)
        MOV     [DI],BX
        MOV     DI,[BP]+10      ;invio valore del potenziometro A(X)
        MOV     [DI],CX
        MOV     DI,[BP]+6       ;invio valore del potenziometro A(X)
        MOV     [DI],DX

        MOV     SP,BP           ;parametri di ritorno
        POP     BP
        RET     10

GIOCO   ENDP                    ;fine della procedura GIOCO
CODICE  ENDS                    ;fine del segmento di codice

        END                     ;fine del programma
```

Figura 10.1 Codice in linguaggio assemblatore per interfacciare il joystick con il programma principale APL

di file, oltre a caratteristiche standard APL, e rappresenta un punto di riferimento per tutte le altre versioni di APL.
APL è un linguaggio interpretato (come il BASIC) e un programma interpretato si differenzia da un programma compilato, in quanto il programmatore non abbandona mai l'interprete per ritornare al sistema. Queste differenze appaiono evidenti se si confronta attentamente il prossimo esempio con quelli che lo seguiranno nel corso di questo capitolo.
Il linguaggio APL della STSC supporta un'istruzione ◻CALL di chiamata alle routine codificate in linguaggio macchina. Questa istruzione presenta la seguente sintassi:

risultato ← ◻CALL *routine*
risultato ← *registri* ◻CALL *routine*

La variabile *risultato* viene restituita dalla routine chiamata, la variabile *registri* referenzia un vettore di interi (da 0 a 7), che identifica l'insieme di registri AX, BX, CX, DX, BP, SI e DI, mentre *routine* è un array non vuoto contenente la routine da eseguire, che deve soddisfare le seguenti condizioni:

1. deve terminare con un'istruzione di ritorno di tipo FAR (0CBH)
2. i registri SS e SP devono essere restituiti immodificati
3. non può utilizzare più di 498 byte di stack
4. deve essere rilocabile
5. è richiesto un offset di 10 byte all'inizio del codice se si usa l'offset per indirizzare le variabili.

Per utilizzare il codice di questa routine, basta memorizzare su file e tradurre in linguaggio macchina il programma di Figura 10.1. Al termine della fase di traduzione, viene prodotto il file .LST indicato in Figura 10.2.
I numeri presenti all'estrema sinistra del listato del file .LST rappresentano i numeri di linea, mentre quelli alla loro destra rappresentano la codifica equivalente in linguaggio macchina dei mnemonici del linguaggio assemblatore. Particolarmente interessante risulta la parte di codice compresa tra l'istruzione CMP DX,0 e la seconda istruzione INT 15H. Non occorre codice addizionale di supporto, in quanto è già definito nell'istruzione ◻CALL del linguaggio APL. La Tabella 10.1 mostra la traduzione in codice macchina espressa nel formato esadecimale e decimale.
Per avere informazioni più dettagliate sull'interfacciamento con il linguaggio macchina, si consulti il manuale del linguaggio APL della STSC sotto la voce System Functions.

```
                ;per 80286/80386
                ;programma che richiede valori di pulsanti o di  potenziometri
                ;dalle leve di comando A & B; con interruzioni BIOS restituisce i
                ;valori ad un programma principale scritto in APL STSC

                        PUBLIC  GIOCO
0000            CODICE  SEGMENT 'CODICE'
                        ASSUME  CS:CODICE
0000            GIOCO   PROC    FAR          ;la procedura si chiama GIOCO
0000 55                 PUSH    BP           ;salva BP sullo stack
0001 8B EC              MOV     BP,SP        ;trasferisce lo stack pointer al registro BP

0003 8B 76 16           MOV     SI,[BP]+22   ;informazione sull'operazione da programma principale
0006 8B 14              MOV     DX,[SI]      ;salva l'informazione nel registro DX
0008 83 FA 00           CMP     DX,0         ;se è zero, continua e legge i pulsanti
000B 75 11              JNE     POT          ;se non è zero,  salta e legge i potenziometri

                ;codice per il campionamento dei quattro pulsanti
000D B4 84              MOV     AH,84H       ;interfaccia leva di comando
000F BA 0000            MOV     DX,0         ;DX=0 per informazione pulsanti
0012 CD 15              INT     15H
0014 B1 04              MOV     CL,04        ;rotazione di quattro bit
0016 D2 C8              ROR     AL,CL        ;bit 7-4 nei bit 3-0
0018 25 000F            AND     AX,0FH       ;isola i quattro bit meno significativi
001B EB 08 90           JMP     SEND         ;restituzione dell'informazione

                ;codice per il campionamento di quattro potenziometri (A(X), A(Y), B(X) e B(Y))
001E B4 84      POT:    MOV    AH,84H        ;interfaccia leva comando
0020 BA 0001            MOV    DX,01H        ;lettura potenziometri
0023 CD 15              INT    15H

                ;codice per restituire i valori dei registri AX, BX, CX e DX.
0025 8B 7E 12   SEND:   MOV    DI,[BP]+18    ;invio valore del potenziometro A(X)
0028 89 05              MOV    [DI],AX
002A 8B 7E 0E           MOV    DI.[BP]+14    ;invio valore del potenziometro A(X)
002D 89 1D              MOV    [DI],BX
002F 8B 7E 0A           MOV    DI,[BP]+10    ;invio valore del potenziometro A(X)
0032 89 0D              MOV    [DI],CX
0034 8B 7E 06           MOV    DI,[BP]+6     ;invio valore del potenziometro A(X)
0037 89 15              MOV    [DI],DX

0039 8B E5              MOV    SP,BP         ;parametri di ritorno
003B 5D                 POP    BP
003C CA 000A            RET    10

003F            GIOCO   ENDP                 ;fine della procedura GIOCO
003F            CODICE  ENDS                 ;fine del segmento di codice

                        END                  ;fine del programma
```

Figura 10.2 Versione tradotta (.LST) del programma in linguaggio assemblatore di interfaccia con il programma APL

Tabella 10.1 Traduzione in codice macchina del programma Adattatore di Gioco in linguaggio APL

Esadecimale	Decimale
83	131
FA	250
00	0
75	117
11	17
B4	180
84	132
BA	186
0000	0
	0
CD	205
15	21
B1	177
04	4
D2	210
C8	200
25	37
000F	15 (prima il byte basso)
	0
EB	235
08	8
90	144
B4	180
84	132
BA	186
0001	1 (prima il byte basso)
	0
CD	205
15	21
(CB)	203 (aggiunto obbligatoriamente)

10.2 Il Turbo Pascal della Borland

Il Pascal è il linguaggio che viene oggi più frequentemente utilizzato nelle università per insegnare la programmazione dei calcolatori. La Borland ha prodotto una versione di compilatore Pascal molto efficiente (insieme ad un editor di video), interfacciabile con l'80287 e con un supporto matematico BCD. La velocità del compilatore rende la programmazione in Turbo Pascal quasi simile alla programmazione in un linguaggio interpretato.

Nel seguente esempio, il joystick viene ripetutamente interrogato fino a quando l'utente preme un tasto. Il programma in linguaggio assemblatore contiene codice di supporto per permettere l'invio e la ricezione di informazioni. La Figura 10.3 mostra il listato completo del programma in linguaggio assemblatore.

```
;per 80286/80386
;programma che richiede valori di pulsanti o di  potenziometri
;dalle leve di comando A & B; con interruzioni BIOS restituisce i
;valori ad un programma principale scritto in Turbo Pascal
;Borland

CODICE  SEGMENT 'CODICE'
        ASSUME  CS:CODICE
GIOCO   PROC    FAR             ;la procedura si chiama GIOCO
        PUSH    BP              ;salva BP sullo stack
        MOV     BP,SP           ;trasferisce lo stack pointer al registro BP

        MOV     SI,[BP]+20      ;informazione sull'operazione da programma principale
        MOV     DX,[SI]         ;salva l'informazione nel registro DX
        CMP     DX,0            ;se e zero, continua e legge i pulsanti
        JNE     POT             ;se non è zero,  salta e legge i potenziometri

;codice per il campionamento dei quattro pulsanti
        MOV     AH,84H          ;interfaccia leva di comando
        MOV     DX,0            ;DX=0 per informazione pulsanti
        INT     15H
        MOV     CL,04           ;rotazione di quattro bit
        ROR     AL,CL           ;bit 7-4 nei bit 3-0
        AND     AX,0FH          ;isola i quattro bit meno significativi
        JMP     SEND            ;restituzione dell'informazione

;codice per il campionamento di quattro potenziometri (A(X), A(Y), B(X) e B(Y))
POT:    MOV     AH,84H          ;interfaccia leva comando
        MOV     DX,01H          ;lettura potenziometri
        INT     15H

;codice per restituire i valori dei registri AX, BX, CX e DX.
SEND:   MOV     DI,[BP]+16      ;invio valore del potenziometro A(X)
        MOV     [DI],AX
        MOV     DI,[BP]+12      ;invio valore del potenziometro A(X)
        MOV     [DI],BX
        MOV     DI,[BP]+8       ;invio valore del potenziometro A(X)
        MOV     [DI],CX
        MOV     DI,[BP]+4       ;invio valore del potenziometro A(X)
        MOV     [DI],DX

        MOV     SP,BP           ;parametri di ritorno
        POP     BP
        RET     10

GIOCO   ENDP                    ;fine della procedura GIOCO
CODICE  ENDS                    ;fine del segmento di codice

        END                     ;fine del programma
```

Figura 10.3 Codice in linguaggio assemblatore che interfaccia il joystick con il programma principale in Turbo Pascal

Riportiamo qui di seguito parte del codice di supporto:

```
;per 80286/80386
;programma che richiede valori di pulsanti o di  potenziometri
;dalle leve di comando A & B; con interruzioni BIOS restituisce i
;valori ad un programma principale scritto in Turbo Pascal
;Borland

CODICE    SEGMENT 'CODICE'
          ASSUME  CS:CODICE
GIOCO     PROC    FAR          ;la procedura si chiama GIOCO
          PUSH    BP           ;salva BP sullo stack
          MOV     BP,SP        ;trasferisce lo stack pointer al registro BP

          MOV     SI,[BP]+20   ;informazione sull'operazione da programma principale
          MOV     DX,[SI]      ;salva l'informazione nel registro DX

(-->il codice di gestione del joystick è stato rimosso da qui<--)

;codice per restituire i valori dei registri AX, BX, CX e DX.
SEND:     MOV   DI,[BP]+16     ;invio valore del potenziometro A(X)
          MOV   [DI],AX
          MOV   DI,[BP]+12     ;invio valore del potenziometro A(X)
          MOV   [DI],BX
          MOV   DI,[BP]+8      ;invio valore del potenziometro A(X)
          MOV   [DI],CX
          MOV   DI,[BP]+4      ;invio valore del potenziometro A(X)
          MOV   [DI],DX

          MOV   SP,BP          ;parametri di ritorno
          POP   BP
          RET   10
```

Il programma in linguaggio assemblatore viene tradotto e convertito in un file .COM, una volta risolti i riferimenti esterni in esso contenuti (i programmi in Turbo Pascal possono infatti essere costituiti da un unico segmento). La procedura GIOCO viene dichiarata di tipo NEAR. Il contenuto di BP (Base Pointer) viene salvato sullo stack, in modo che il valore del registro SP (Stack Pointer) possa essere trasferito al registro BP. Generalmente, il passaggio dei parametri avviene tramite lo stack. Le variabili possono essere trasferite dal programma principale al programma in linguaggio assemblatore attraverso il registro SI (Indice Sorgente) e, viceversa, attraverso il registro DI (Indice Destinazione). In questo esempio, cinque variabili di tipo word (DW) vengono trasferite tra i due programmi. Poiché GIOCO è una procedura di tipo NEAR, l'indirizzo di ritorno occupa i primi quattro byte dello stack. L'indirizzo della variabile del programma principale (BY) si trova in [BP]+4 e viene trasferito nel registro DI. Il valore intero contenuto nel registro DX viene poi trasferito in quella locazione di memoria con l'istruzione MOV [DI],DX. Avanzando di altri quattro byte sullo stack, si trova l'indirizzo della variabile successiva, e così via. In questo modo, vengono restituiti al programma principale quattro valori interi, che rappresentano le letture del potenziometro del joystick, supponendo che inizialmente il registro DX

sia stato inizializzato a 1. Se DX invece contiene il valore 0, solo il valore restituito al registro AX ha significato e corrisponde all'informazione relativa ai pulsanti.
La locazione di memoria [BP]+20 referenzia una variabile che viene trasferita dal programma principale al programma scritto in linguaggio assemblatore. In questo modo, il registro DX può essere inizializzato a 1 (lettura potenziometro) o a 0 (lettura pulsanti) dal programma chiamante. L'indirizzo contenuto in [BP]+20 viene trasferito nel registro SI. Il valore intero referenziato tramite SI viene poi trasferito nel registro DX. Utilizzando la stessa tecnica, è possibile realizzare un passaggio illimitato di parametri.
Prima di abbandonare questa routine, viene ripristinato il contenuto del registro BP e viene eseguita l'istruzione di ritorno RET di tipo NEAR. L'operando 10 dell'istruzione RET indica che lo stack deve essere svuotato dei parametri in esso contenuti (10 byte di memoria), che erano stati passati alla routine dal programma chiamante.
La Figura 10.4 mostra il programma principale in Turbo Pascal. Si tenga presente che il programma in linguaggio assemblatore GIOCO.COM è contenuto nel drive B. Altra considerazione da fare riguarda l'ordine con cui le variabili vengono passate. OPER, AX, AY, BX e BY sono variabili intere: OPER passa al programma in linguaggio assemblatore (registro DX) il valore 1 o 0 e il suo indirizzo è [BP]+20, mentre AX, AY, BX e BY costituiscono i parametri di ritorno al programma principale. Si tenga presente che l'indirizzo di BY è [BP]+4.
Il programma in linguaggio assemblatore deve essere scritto, tradotto in codice macchina, devono essere risolti i suoi riferimenti esterni e deve essere

```
PROGRAM GIOCOTP;

VAR
  OPER,AX,AY,BX,BY:INTEGER;

PROCEDURE REP(VAR OPER,AX,AY;BX,BY:INTEGER); EXTERNAL 'B:GIOCO.COM';

PROCEDURE INFORM;
BEGIN
  OPER:=1;
  REP(OPER,AX,AY,BX,BY);
  WRITELN(AX,'     ',AY,'     ',BX,'     ',BY,'     ')
END;

BEGIN {codice principale}
WHILE NOT KEYPRESSED DO
    INFORM
END.
```

Figura 10.4 Programma in Turbo Pascal che chiama il programma in linguaggio assemblatore per campionare la porta del joystick

convertito in un file .COM utilizzando l'utilità EXE2BIN. Riportiamo qui di seguito l'immagine che viene visualizzata in uscita sullo schermo.

```
B>C:MASM GIOCO;
IBM Personal Computer MACRO Assembler  Version 2.00
(C)Copyright IBM Corp 1981, 1984
(C)Copyright Microsoft Corp 1981, 1983, 1984

50096 Bytes free

Warning Severe
Errors  Errors
0       0


B>C:LINK GIOCO,GIOCO,NUL,;
IBM Personal Computer Linker
Version 2.30 (C) Copyright IBM Corp. 1981, 1985

Warning: no stack segment

B:

B>C:EXE2BIN GIOCO GIOCO.COM
```

Il prossimo passo da compiere consiste nel codificare con l'editor Turbo il programma principale di Figura 10.4. All'uscita dall'editor, selezionate l'opzione O e specificate di volere la versione eseguibile del programma principale nel formato .COM.
Uscite quindi dall'ambiente Turbo Pascal ed eseguite il programma. Poiché il compilatore crea a sua volta un file .COM, il programma è in realtà molto piccolo. Si tenga presente che entrambi i file, contenenti i programmi in Pascal e in linguaggio assemblatore, devono essere disponibili al momento dell'esecuzione, ma non devono essere collegati insieme per avere un unico programma funzionante.

10.3 Il BASIC della Microsoft

QuickBASIC della Microsoft rappresenta uno dei compilatori BASIC più completi, in quanto supporta quasi tutte le istruzioni del BASIC, incluse le istruzioni grafiche. QuickBASIC è il compilatore ideale per gli utenti occasionali e rivaleggia sotto diversi aspetti con i compilatori più costosi. Le tecniche di interfaccia che vengono presentate in questo paragrafo si adattano alla maggior parte dei compilatori BASIC che sono disponibili per la famiglia dei PC IBM.
Per semplificare l'esempio di interfaccia, il programma BASIC continua a richiamare il programma in linguaggio assemblatore. Per interrompere l'e-

```
;per 80286/80386
;programma che richiede valori di pulsanti o di  potenziometri
;dalle leve di comando A & B; con interruzioni BIOS restituisce i
;valori ad un programma principale scritto in QuickBASIC
;Microsoft

        PUBLIC  GIOCOB
CODICE  SEGMENT 'CODICE'
        ASSUME  CS:CODICE
GIOCOB  PROC    FAR             ;la procedura si chiama GIOCOB
        PUSH    BP              ;salva BP sullo stack
        MOV     BP,SP           ;trasferisce lo stack pointer al registro BP

        MOV     SI,[BP]+16      ;informazione sull'operazione da programma principale
        MOV     DX,[SI]         ;salva l'informazione nel registro DX
        CMP     DX,0            ;se è zero, continua e legge i pulsanti
        JNE     POT             ;se non è zero,  salta e legge i potenziometri

;codice per il campionamento dei quattro pulsanti
        MOV     AH,84H          ;interfaccia leva di comando
        MOV     DX,0            ;DX=0 per informazione pulsanti
        INT     15H
        MOV     CL,04           ;rotazione di quattro bit
        ROR     AL,CL           ;bit 7-4 nei bit 3-0
        AND     AX,0FH          ;isola i quattro bit meno significativi
        JMP     SEND            ;restituzione dell'informazione

;codice per il campionamento di quattro potenziometri (A(X), A(Y), B(X) e B(Y))
POT:    MOV     AH,84H          ;interfaccia leva comando
        MOV     DX,01H          ;lettura potenziometri
        INT     15H

;codice per restituire i valori dei registri AX, BX, CX e DX.
SEND:   MOV     DI,[BP]+14      ;invio valore del potenziometro A(X)
        MOV     [DI],AX
        MOV     DI,[BP]+12      ;invio valore del potenziometro A(X)
        MOV     [DI],BX
        MOV     DI,[BP]+10      ;invio valore del potenziometro A(X)
        MOV     [DI],CX
        MOV     DI,[BP]+8       ;invio valore del potenziometro A(X)
        MOV     [DI],DX

        MOV     SP,BP           ;parametri di ritorno
        POP     BP
        RET     10

GIOCOB  ENDP                    ;fine della procedura GIOCOB
CODICE  ENDS                    ;fine del segmento di codice

        END                     ;fine del programma
```

Figura 10.5 Codice in linguaggio assemblatore per interfacciare il joystick con il programma principale scritto in QuickBASIC

secuzione, è quindi necessario premere CTRL-ALT-DEL. Il programma in linguaggio assemblatore contiene il codice di supporto che permette l'invio e la ricezione di informazioni con il programma di alto livello. La Figura 10.5 mostra il listato completo del programma in linguaggio assemblatore. Riportiamo qui di seguito una parte del codice di supporto:

```
        PUBLIC  GIOCOB
CODICE  SEGMENT 'CODICE'
        ASSUME  CS:CODICE
GIOCOB  PROC    FAR             ;la procedura si chiama GIOCOB
        PUSH    BP              ;salva BP sullo stack
        MOV     BP,SP           ;trasferisce lo stack pointer al registro BP

        MOV     SI,[BP]+16      ;informazione sull'operazione da programma principale
        MOV     DX,[SI]         ;salva l'informazione nel registro DX

(-->il codice di gestione del joystick è stato rimosso da qui<--)

;codice per restituire i valori dei registri AX, BX, CX e DX.
SEND:   MOV     DI,[BP]+14      ;invio valore del potenziometro A(X)
        MOV     [DI],AX
        MOV     DI,[BP]+12      ;invio valore del potenziometro A(X)
        MOV     [DI],BX
        MOV     DI,[BP]+10      ;invio valore del potenziometro A(X)
        MOV     [DI],CX
        MOV     DI,[BP]+8       ;invio valore del potenziometro A(X)
        MOV     [DI],DX

        MOV     SP,BP           ;parametri di ritorno
        POP     BP
        RET     10
```

Questa routine in linguaggio assemblatore viene tradotta in codice macchina e viene collegata mediante il linker con la versione compilata del programma scritto in BASIC. Le routine in linguaggio assemblatore che vengono collegate con i programmi BASIC possono essere costituite da più di un segmento. La procedura che viene invocata dalla routine BASIC deve essere dichiarata PUBLIC per rendere possibile il passaggio dei parametri con il programma principale. La procedura GIOCOB deve essere di tipo FAR, per cui necessita – al termine della sua codifica – di una istruzione di ritorno RET di tipo FAR. Il contenuto di BP (Base Pointer) viene salvato sullo stack, in modo che il valore del registro SP (Stack Pointer) possa essere trasferito al registro BP. Generalmente, il passaggio dei parametri avviene tramite lo stack. Le variabili possono essere trasferite dal programma principale al programma scritto in linguaggio assemblatore attraverso il registro SI (Indice Sorgente) e, viceversa, attraverso il registro DI (Indice Destinazione). In questo esempio, cinque variabili di tipo word (DW) vengono trasferite tra i due programmi. Poiché GIOCOB è una procedura di tipo FAR, l'indirizzo di ritorno occupa i primi otto byte dello stack. L'indirizzo della variabile del programma principale (BY) si trova in [BP]+8 e viene trasferito nel registro DI. Il valore intero contenuto nel registro DX viene poi trasferito in quella loca-

zione di memoria con l'istruzione MOV [DI],DX. Avanzando di altri due byte sullo stack, si trova l'indirizzo della variabile successiva, e così via. In questo modo, vengono restituiti al programma principale quattro valori interi, che rappresentano le letture del potenziometro del joystick, supponendo che inizialmente il registro DX sia stato inizializzato a 1. Se DX invece contiene il valore 0, solo il valore restituito al registro AX ha significato e corrisponde all'informazione relativa ai pulsanti.
La locazione di memoria [BP]+16 referenzia una variabile che viene trasferita dal programma principale al programma scritto in linguaggio assemblatore. In questo modo, il registro DX può essere inizializzato a 1 (lettura potenziometro) o a 0 (lettura pulsanti) dal programma chiamante. L'indirizzo contenuto in [BP]+16 viene trasferito nel registro SI. Il valore intero referenziato tramite SI viene poi trasferito nel registro DX. Utilizzando la stessa tecnica, è possibile realizzare un passaggio illimitato di parametri.
Prima di abbandonare questa routine, viene ripristinato il contenuto del registro BP e viene eseguita l'istruzione di ritorno RET di tipo FAR. L'operando 10 dell'istruzione RET indica che lo stack deve essere svuotato dei parametri in esso contenuti (10 byte di memoria), che erano stati passati alla routine dal programma chiamante.
La Figura 10.6 mostra il programma principale in BASIC. Si tenga presente che il programma in linguaggio assemblatore viene invocato utilizzando l'istruzione di chiamata ad una procedura esterna. Questo programma si chiama GIOCOB.ASM e, insieme alla versione in codice oggetto .OBJ, è contenuto nel drive B. Un'altra considerazione da fare riguarda l'ordine con cui le variabili vengono passate. OPER%, AX%, AY%, BX% e BY% sono variabili intere: OPER% passa al programma in linguaggio assemblatore (registro DX) il valore 1 o 0 e il suo indirizzo è [BP]+16, mentre AX%, AY%, BX% e BY% costituiscono i parametri di ritorno al programma principale. Si tenga presente che l'indirizzo di BY è [BP]+8.
Il programma in linguaggio assemblatore deve essere scritto e tradotto in codice macchina. Riportiamo qui di seguito ciò che viene visualizzato sullo schermo:

```
B>C:MASM GIOCOB.ASM;
IBM Personal Computer MACRO Assembler  Version 2.00
```

```
LET OPER%=1
FOR I=1 TO 200
  CALL GIOCOB(OPER%,AX%,BX%,CX%,DX%)
  PRINT AX%,BX%,CX%,DX%
NEXT I
END
```

Figura 10.6 Programma in QuickBASIC che chiama il programma in linguaggio assemblatore per campionare la porta del joystick

```
(C)Copyright IBM Corp 1981, 1984
(C)Copyright Microsoft Corp 1981, 1983, 1984

50096 Bytes free

Warning Severe
Errors  Errors
0       0
```

Il prossimo passo da compiere è quello di scrivere il programma in BASIC con un editor (come ad esempio l'editor di video di Peter Norton). Successivamente, il programma BASIC viene compilato utilizzando i seguenti comandi:

```
C:BASCOM B:GIOCOBMS/O,B:,,;
Microsoft Quick BASIC Compiler
Version 1.00
(C) Copyright Microsoft Corp. 1982, 1983, 1984, 1985


49158 Bytes Available
48704 Bytes Free

      0 Warning Error(s)
      0 Severe  Error(s)
```

Abbiamo utilizzato l'opzione /O per rendere eseguibile il programma, senza ricorrere al file BASRUN.
Da ultimo, il programma principale e il programma in linguaggio assemblatore vengono collegati insieme utilizzando i seguenti comandi di sistema (per il Linker del PC IBM, versione 2.30):

```
C:LINK B:GIOCOBMS+B:GIOCOB,B:,,C:;
```

Il risultato della compilazione del programma principale in BASIC è GIOCOBMS.OBJ e risiede nel dischetto presente nel drive B, dove risiede anche la versione tradotta GIOCOB.OBJ del programma in linguaggio assemblatore. Questi programmi in codice oggetto vengono collegati insieme con l'opzione (+) del linker e, dopo questa operazione, non è più necessario mantenere il programma in linguaggio assemblatore sul dischetto. Il programma principale GIOCOBMS.EXE rappresenta la versione eseguibile finale.

10.4 Il C della Microsoft

Il C è stato indicato da molti come il linguaggio di programmazione adatto ad ogni esigenza. Inizialmente è stato impiegato soprattutto nella scrittura di sistemi operativi, ma successivamente ha ampliato il suo campo di azio-

ne dalla gestione di database a programmi precedentemente scritti in assembler. Il C svolge la funzione di "trait d'union" tra i linguaggi di alto livello e il codice assemblatore e, trattandosi di un linguaggio che non ha subito modifiche e ampliamenti nel corso degli anni, il compilatore è semplice e di piccole dimensioni.
Il compilatore C della Microsoft – a partire dalla versione 3.0 – è considerato lo standard tra i compilatori C per piccoli sistemi e viene venduto anche sotto il marchio IBM.
La Microsoft ha utilizzato il linguaggio C per riscrivere il Macro Assembler, che, a partire dalla versione 4.0, esegue infatti ad una velocità tre volte superiore rispetto alla versione 3.0 del Macro Assembler Microsoft e alla versione 2.0 del Macro Assembler IBM. Questo testimonia la notevole velocità di esecuzione e la compattezza di codice dei programmi scritti in C (di solito, nessun linguaggio supera in velocità il linguaggio assemblatore).
Il programma di interfaccia che presentiamo in questo paragrafo, interroga ripetutamente il joystick fino a quando non viene eseguito il comando CTRL-ALT-DEL. Il programma in C visualizza sullo schermo i risultati, mentre il programma in linguaggio assemblatore contiene il codice di supporto per permettere l'invio e la ricezione delle informazioni con il programma principale.
La Figura 10.7 mostra il programma in linguaggio assemblatore nella sua completezza.
Riportiamo qui di seguito una parte del codice di supporto:

```
        public  _giococ         ;dichiarazioni C indispensabili
    _testo  segment byte public 'codice'
        assume  cs:_testo
    _giococ proc    near          ;la procedura si chiama GIOCOC
        push    bp              ;salva BP sullo stack
        mov     bp,sp           ;trasferisce lo stack pointer al registro BP
        push    di              ;salva il contenuto originale di DI e SI
        push    si

        mov     si,[bp+4]       ;informazione sull'operazione da programma principale
        mov     dx,[si]         ;salva l'informazione nel registro DX
        cmp     dx,0            ;se è zero, continua e legge i pulsanti
        jne     pot             ;se non è zero,  salta e legge i potenziometri

  (-->il codice di gestione del joystick è stato rimosso da qui<--)

    send:   mov   di,[bp+6]       ;invio valore del potenziometro A(X)
        mov   [di],ax
        mov   di,[bp+8]         ;invio valore del potenziometro A(X)
        mov   [di],bx
        mov   di,[bp+10]        ;invio valore del potenziometro A(X)
        mov   [di],cx
        mov   di,[bp+12]        ;invio valore del potenziometro A(X)
        mov,  [di],dx

        pop   si                ;parametri di ritorno
        pop   di
        mov   sp,bp
        pop   bp
        ret
```

```
;per 80286/80386
;programma che richiede valori di pulsante o di  potenziometro
;dalle leve di comando A & B; con interruzioni BIOS restituisce i
;valori ad un programma principale scritto in C Microsoft

        public  _giococ       ;dichiarazioni C indispensabili
_testo  segment byte public 'codice'
        assume  cs:_testo
_giococ proc    near          ;la procedura si chiama GIOCOC
        push    bp            ;salva BP sullo stack
        mov     bp,sp         ;trasferisce lo stack pointer al registro BP
        push    di            ;salva il contenuto originale di DI e SI
        push    si

        mov     si,[bp+4]     ;informazione sull'operazione da programma principale
        mov     dx,[si]       ;salva l'informazione nel registro DX
        cmp     dx,0          ;se è zero, continua e legge i pulsanti
        jne     pot           ;se non è zero,  salta e legge i potenziometri

;codice per il campionamento dei quattro pulsanti
        mov     ah,84H        ;interfaccia leva di comando
        mov     dx,0          ;DX=0 per informazione pulsanti
        int     15h
        mov     cl,04         ;rotazione di quattro bit
        ror     al,cl         ;bit 7-4 in bit 3-0
        and     ax,0fH        ;isola i quattro bit meno significativi
        jmp     send          ;restituzione dell'informazione

;codice per il campionamento di quattro potenziometri (A(X), A(Y), B(X) e B(Y))
pot:    mov     AH,84H        ;interfaccia leva comando
        mov     dx,01H        ;lettura potenziometri
        int     15H

;codice per restituire i valori dei registri AX, BX, CX e DX.
send:   mov     di,[bp+6]     ;invio valore del potenziometro A(X)
        mov     [di],ax
        mov     di,[bp+8]     ;invio valore del potenziometro A(X)
        mov     [di],bx
        mov     di,[bp+10]    ;invio valore del potenziometro A(X)
        mov     [di],cx
        mov     di,[bp+12]    ;invio valore del potenziometro A(X)
        mov     [di],dx

        pop     si            ;parametri di ritorno
        pop     di
        mov     sp,bp
        pop     bp
        ret

_giococ endp                  ;fine della procedura GIOCOB
_testo  ends                  ;fine del segmento di codice

        end                   ;fine del programma
```

Figura 10.7 Codice in linguaggio assemblatore per interfacciare il joystick con il programma principale scritto in C

Il codice in linguaggio assemblatore viene tradotto e collegato con la versione compilata del programma scritto in C. Vediamo alcune caratteristiche interessanti del programma in linguaggio assemblatore. Innanzitutto, è scritto in minuscolo, poiché il compilatore C differenzia lettere minuscole e maiuscole. Inoltre, la definizione di segmento e di procedura inizia con il carattere di sottolineatura (questo è un requisito di ogni programma di interfaccia). La procedura giococ viene dichiarata public e di tipo NEAR, per cui la sua ultima istruzione è una RET di tipo NEAR. Il contenuto di BP (Base Pointer) viene salvato sullo stack, in modo che il valore del registro SP (Stack Pointer) possa essere trasferito al registro BP. Generalmente, il passaggio dei parametri avviene tramite lo stack. Le variabili possono essere trasferite dal programma principale al programma scritto in linguaggio assemblatore attraverso il registro SI (Indice Sorgente) e, viceversa, attraverso il registro DI (Indice Destinazione). Se i registri SI e DI vengono utilizzati dal programma in linguaggio assemblatore, i valori originali in essi contenuti devono essere salvati sullo stack e ripristinati prima di uscire dal programma. In questo esempio, cinque variabili di tipo word (DW) vengono trasferite tra i due programmi. Poiché giococ è una procedura di tipo NEAR, l'indirizzo di ritorno occupa i primi quattro byte dello stack. L'indirizzo della variabile del programma principale (&oper) si trova in [BP]+4 e viene trasferito nel registro SI. Il valore intero contenuto nel registro SI viene poi trasferito nel registro DX con l'istruzione MOV dx,[si]. Avanzando di altri due byte sullo stack, si trova l'indirizzo della variabile successiva (&ax), e così via. In questo modo, vengono restituiti al programma principale quattro valori interi, che rappresentano le letture del potenziometro del joystick, supponendo che inizialmente il registro DX sia stato inizializzato a 1. Se DX invece contiene il valore 0, solo il valore restituito al registro AX ha significato e corrisponde all'informazione relativa ai pulsanti. Utilizzando la stessa tecnica, è possibile realizzare un passaggio illimitato di parametri.
Prima di abbandonare questa routine, viene ripristinato il contenuto dei registri SI, DI e BP e viene eseguita l'istruzione di ritorno RET di tipo NEAR. La Figura 10.8 mostra il programma principale in C. Si tenga presente che il programma in linguaggio assemblatore viene invocato utilizzando l'istruzione di chiamata ad una procedura esterna e che la chiamata viene eseguita senza il carattere di sottolineatura che precede 'giococ'. Questo programma si chiama GIOCOC.ASM e, insieme alla versione in codice oggetto .OBJ, è contenuto nel drive B. Un'altra considerazione da fare riguarda l'ordine con cui le variabili vengono passate; oper%, ax%, ay%, bx% e by% sono variabili intere: oper% passa al programma in linguaggio assemblatore (registro DX) il valore 1 o 0 e il suo indirizzo è [BP]+16, mentre ax%, ay%, bx% e by% costituiscono i parametri di ritorno al programma principale. Si tenga presente che l'indirizzo di by è [BP]+12.
I passi necessari per creare l'esempio di interfaccia con il programma C sono abbastanza semplici. Innanzitutto, viene scritto il programma in linguag-

```
int oper, ax, ay, bx, by;

main()
{
        extern giococ();
        oper = 1;
repeat: giococ(&oper,&ax,&ay,&bx,&by);
        printf("%d %d %d %d\n", ax, ay, bx, by);
        goto repeat;
}
```

Figura 10.8 Programma in C che chiama il programma in linguaggio assemblatore per campionare la porta del joystick

gio assemblatore con un editor di video (ad esempio l'editor di Norton). Poi, il programma viene tradotto in codice macchina, nel modo seguente:

```
B>C:MASM GIOCOC.ASM;
IBM Personal Computer MACRO Assembler  Version 2.00
(C)Copyright IBM Corp 1981, 1984
(C)Copyright Microsoft Corp 1981, 1983, 1984

50096 Bytes free

Warning Severe
Errors  Errors
0       0
```

In questa fase viene creato il file .OBJ sul dischetto che risiede nel drive B. Successivamente, viene scritto il programma in C con lo stesso editor e viene compilato con il compilatore C per creare un file di codice oggetto .OBJ, utilizzando il seguente comando (per compilatore C della Microsoft, versione 3.0):

```
C>MSC B:GIOCOCMS,B:,/G2/FPc87;
```

Da ultimo, i due file .OBJ devono essere collegati insieme, utilizzando il Linker di sistema (per Linker Microsoft 8086, versione 3.01):

```
C>LINK B:GIOCOCMS+B:GIOCOC,B:,,C:;
```

In questo esempio, la versione compilata del programma principale in C (GIOCOCMS) risiede nel dischetto presente nel drive B, dove risiede anche la versione in codice oggetto GIOCOC.OBJ del programma scritto in linguaggio assemblatore. Questi programmi vengono collegati insieme utilizzando l'opzione (+) del Linker. Conclusa questa operazione, non è più necessario man-

tenere su dischetto il programma in linguaggio assemblatore. Il programma principale GIOCOCMS.EXE costituisce la versione eseguibile finale.

10.5 Il FORTRAN della IBM

Il linguaggio di programmazione FORTRAN è particolarmente adatto per elaborazioni di tipo numerico. Anche se la sua importanza è diminuita con l'avvento di linguaggi come Ada, Pascal e C, il FORTRAN continua ad essere molto noto. Il compilatore FORTRAN della IBM viene considerato lo standard tra i compilatori FORTRAN per piccoli sistemi.
Nel prossimo esempio, il programma FORTRAN interroga ripetutamente il programma scritto in linguaggio assemblatore. Per interrompere l'esecuzione, è dunque necessario dare il comando CTRL-ALT-DEL. Il programma in linguaggio assemblatore contiene codice di supporto per permettere l'invio e la ricezione di informazioni con il programma principale. La Figura 10.9 mostra un listato completo del programma in linguaggio assemblatore. Riportiamo qui di seguito una parte del codice di supporto:

```
     PUBLIC  GIOCOB
CODICE SEGMENT 'CODICE'
     ASSUME  CS:CODICE
GIOCOF PROC   FAR          ;la procedura si chiama GIOCOB
     PUSH    BP           ;salva BP sullo stack
     MOV     BP,SP        ;trasferisce lo stack pointer al registro BP

     MOV     SI,[BP]+22   ;informazione sull'operazione da programma principale
     MOV     DX,[SI]      ;salva l'informazione nel registro DX

(-->il codice di gestione del joystick è stato rimosso da qui<--)

;codice per restituire i valori dei registri AX, BX, CX e DX.
SEND:  MOV   DI,[BP]+18    ;invio valore del potenziometro A(X)
     MOV     [DI],AX
     MOV     DI,[BP]+14   ;invio valore del potenziometro A(X)
     MOV     [DI],BX
     MOV     DI,[BP]+10   ;invio valore del potenziometro A(X)
     MOV     [DI],CX
     MOV     DI,[BP]+6    ;invio valore del potenziometro A(X)
     MOV     [DI],DX

     MOV     SP,BP        ;parametri di ritorno
     POP     BP
     RET     10
```

Questa routine in linguaggio assemblatore viene tradotta in codice macchina e viene collegata con la versione compilata del programma scritto in FORTRAN. La procedura GIOCOF deve essere dichiarata PUBLIC per rendere possibile il passaggio dei parametri con il programma principale e deve essere di tipo FAR, per cui necessita – al termine della sua codifica – di una

```
;per 80286/80386
;programma che richiede valori di pulsanti o di  potenziometri
;dalle leve di comando A & B; con interruzioni BIOS restituisce i
;valori ad un programma principale scritto in FORTRAN IBM

        PUBLIC  GIOCOB
CODICE  SEGMENT 'CODICE'
        ASSUME  CS:CODICE
GIOCOF  PROC    FAR         ;la procedura si chiama GIOCOB
        PUSH    BP          ;salva BP sullo stack
        MOV     BP,SP       ;trasferisce lo stack pointer al registro BP

        MOV     SI,[BP]+22  ;informazione sull'operazione da programma principale
        MOV     DX,[SI]     ;salva l'informazione nel registro DX
        CMP     DX,0        ;se è zero, continua e legge i pulsanti
        JNE     POT         ;se non è zero,  salta e legge i potenziometri

;codice per il campionamento dei quattro pulsanti
        MOV     AH,84H      ;interfaccia leva di comando
        MOV     DX,0        ;DX=0 per informazione pulsanti
        INT     15H
        MOV     CL,04       ;rotazione di quattro bit
        ROR     AL,CL       ;bit 7-4 nei bit 3-0
        AND     AX,0FH      ;isola i quattro bit meno significativi
        JMP     SEND        ;restituzione dell'informazione

;codice per il campionamento di quattro potenziometri (A(X), A(Y), B(X) e B(Y))
POT:    MOV   AH,84H        ;interfaccia leva comando
        MOV   DX,01H        ;lettura potenziometri
        INT   15H

;codice per restituire i valori dei registri AX, BX, CX e DX.
SEND:   MOV   DI,[BP]+18    ;invio valore del potenziometro A(X)
        MOV   [DI],AX
        MOV   DI,[BP]+14    ;invio valore del potenziometro A(X)
        MOV   [DI],BX
        MOV   DI,[BP]+10    ;invio valore del potenziometro A(X)
        MOV   [DI],CX
        MOV   DI,[BP]+6     ;invio valore del potenziometro A(X)
        MOV   [DI],DX

        MOV   SP,BP         ;parametri di ritorno
        POP   BP
        RET   10

GIOCOB  ENDP                ;fine della procedura GIOCOB
CODICE  ENDS                ;fine del segmento di codice

        END                 ;fine del programma
```

Figura 10.9 Codice in linguaggio assemblatore per interfacciare il joystick con il programma principale scritto in FORTRAN

istruzione di ritorno RET di tipo FAR. Il contenuto di BP (Base Pointer) viene salvato sullo stack, in modo che il valore del registro SP (Stack Pointer) possa essere trasferito al registro BP. Generalmente, il passaggio dei parametri avviene tramite lo stack. Le variabili possono essere trasferite dal programma principale al programma scritto in linguaggio assemblatore attraverso il registro SI (Indice Sorgente) e, viceversa, attraverso il registro DI (Indice Destinazione). In questo esempio, cinque variabili di tipo word (DW) vengono trasferite tra i due programmi. Poiché GIOCOF è una procedura di tipo FAR, l'indirizzo di ritorno occupa i primi otto byte dello stack. L'indirizzo della variabile del programma principale (BY) si trova in [BP]+6 e viene trasferito nel registro DI. Il valore intero contenuto nel registro DX viene poi trasferito in quella locazione di memoria con l'istruzione MOV [DI],DX. Avanzando di altri due byte sullo stack, si trova l'indirizzo della variabile successiva (BX), e così via. In questo modo, vengono restituiti al programma principale quattro valori interi, che rappresentano le letture del potenziometro del joystick, supponendo che inizialmente il registro DX sia stato inizializzato a 1. Se DX invece contiene il valore 0, solo il valore restituito al registro AX ha significato e corrisponde all'informazione relativa ai pulsanti. Utilizzando la stessa tecnica, è possibile realizzare un passaggio illimitato di parametri.
Prima di abbandonare questa routine, viene ripristinato il contenuto del registro BP e viene eseguita l'istruzione di ritorno RET di tipo FAR. L'operando 10 dell'istruzione RET indica che lo stack deve essere svuotato dei parametri in esso contenuti (10 byte di memoria), che erano stati passati alla routine dal programma chiamante.
La Figura 10.10 mostra il programma principale in FORTRAN. Si tenga presente che il programma in linguaggio assemblatore viene invocato utilizzando l'istruzione di chiamata ad una procedura esterna. Questo programma si chiama GIOCOF.ASM e, insieme alla versione in codice oggetto .OBJ, è contenuto nel drive B. Un'altra considerazione da fare riguarda l'ordine con cui le variabili vengono passate. OPER, AX, AY, BX e BY sono variabili inte-

```
      INTEGER OPER,AX,AY,BX,BY
      OPER=1
      DO 3 I=1,500
1     CALL GIOCOF(OPER,AX,AY,BX,BY)
      WRITE(*,2)AX,AY,BX,BY
2     FORMAT (4I5)
3     CONTINUE
      END
```

Figura 10.10 Programma in FORTRAN che chiama il programma in linguaggio assemblatore per campionare la porta del joystick

re: OPER passa al programma in linguaggio assemblatore (registro DX) il valore 1 o 0 e il suo indirizzo è [BP]+22, mentre AX, AY, BX e BY costituiscono i parametri di ritorno al programma principale. Si tenga presente che l'indirizzo di BY è [BP]+6.
I passi necessari per creare l'esempio di interfaccia con il programma FORTRAN sono abbastanza semplici. Innanzitutto, viene scritto il programma in linguaggio assemblatore con un editor di video (ad esempio l'editor di Norton). Poi, il programma viene tradotto in codice macchina, nel modo seguente:

```
B>C:MASM GIOCOF.ASM;
IBM Personal Computer MACRO Assembler  Version 2.00
(C)Copyright IBM Corp 1981, 1984
(C)Copyright Microsoft Corp 1981, 1983, 1984

50096 Bytes free

Warning Severe
Errors  Errors
0       0
```

In questa fase viene creato il file .OBJ sul dischetto che risiede nel drive B. Successivamente, viene scritto il programma in FORTRAN con lo stesso editor e viene compilato con il compilatore FORTRAN per creare un file di codice oggetto .OBJ, utilizzando il seguente comando:

```
A>FOR1 B:GIOCOFMS,B:GIOCOFMS,NUL,NUL;

IBM Personal Computer FORTRAN Compiler
Versione 2.00
(C)Copyright IBM Corp 1982, 1984
(C)Copyright Microsoft Corp 1982, 1984

 Pass One     No Errors Detected
                 8 Source Lines

A>FOR2

 Code Area Size = #00D1   (  209)
 Cons Area Size = #000C   (   12)
 Data Area Size = #0021   (   33)

 Pass Two     No Errors Detected
```

Da ultimo, i due file .OBJ devono essere collegati insieme, utilizzando il Linker di sistema (per Linker Microsoft 8086, versione 3.01):

```
A>LINK B:GIOCOFMS+B:GIOCOF,B:,,A:;
```

In questo esempio, la versione compilata del programma principale in FORTRAN (GIOCOFMS) risiede nel dischetto presente nel drive B, dove risiede anche la versione in codice oggetto GIOCOF.OBJ del programma scritto in linguaggio assemblatore. Questi programmi vengono collegati insieme uti-

lizzando l'opzione (+) del Linker. Conclusa questa operazione, non è più necessario mantenere su dischetto il programma in linguaggio assemblatore. Il programma principale GIOCOFMS.EXE costituisce la versione eseguibile finale.

10.6 Il Pascal della IBM

Il compilatore Pascal della IBM non risulta efficiente come il compilatore Turbo Pascal della Borland. Si tratta, infatti, di un compilatore più costoso, non possiede un editor e impiega un tempo di compilazione estremamente elevato. Il compilatore Pascal della IBM, comunque, potrebbe essere scelto da un programmatore Pascal esperto (l'ultima versione supporta il coprocessore 80287) e crea file in codice oggetto .OBJ che possono essere collegati con programmi multisegmento scritti in linguaggio assemblatore. Questa possibilità viene negata con il Turbo Pascal della Borland.
Nel prossimo esempio, viene interrogato ripetutamente il joystick fino a quando viene premuto un tasto. Il programma in linguaggio assemblatore contiene codice di supporto per permettere l'invio e la ricezione di informazioni. La Figura 10.11 mostra un listato completo del programma in linguaggio assemblatore.
Riportiamo qui di seguito una parte del codice di supporto:

```
        PUBLIC  GIOCOB
CODICE SEGMENT 'CODICE'
        ASSUME  CS:CODICE
GIOCOP PROC     FAR             ;la procedura si chiama GIOCOB
        PUSH    BP              ;salva BP sullo stack
        MOV     BP,SP           ;trasferisce lo stack pointer al registro BP

        MOV     SI,[BP]+14      ;informazione sull'operazione da programma principale
        MOV     DX,[SI]         ;salva l'informazione nel registro DX

(-->il codice di gestione del joystick è stato rimosso da qui<--)

;codice per restituire i valori dei registri AX, BX, CX e DX.
SEND:   MOV     DI,[BP]+12      ;invio valore del potenziometro A(X)
        MOV     [DI],AX
        MOV     DI,[BP]+10      ;invio valore del potenziometro A(X)
        MOV     [DI],BX
        MOV     DI,[BP]+8       ;invio valore del potenziometro A(X)
        MOV     [DI],CX
        MOV     DI,[BP]+6       ;invio valore del potenziometro A(X)
        MOV     [DI],DX

        MOV     SP,BP           ;parametri di ritorno
        POP     BP
        RET     10
```

Questa routine in linguaggio assemblatore viene tradotta in codice macchina e viene collegata con la versione compilata del programma scritto in Pa-

```
;per 80286/80386
;programma che richiede valori di pulsanti o di  potenziometri
;dalle leve di comando A & B; con interruzioni BIOS restituisce i
;valori ad un programma principale scritto in Pascal IBM

        PUBLIC  GIOCOB
CODICE  SEGMENT 'CODICE'
        ASSUME  CS:CODICE
GIOCOP  PROC    FAR           ;la procedura si chiama GIOCOB
        PUSH    BP            ;salva BP sullo stack
        MOV     BP,SP         ;trasferisce lo stack pointer al registro BP

        MOV     SI,[BP]+14    ;informazione sull'operazione da programma principale
        MOV     DX,[SI]       ;salva l'informazione nel registro DX
        CMP     DX,0          ;se è zero, continua e legge i pulsanti
        JNE     POT           ;se non è zero,  salta e legge i potenziometri

;codice per il campionamento dei quattro pulsanti
        MOV     AH,84H        ;interfaccia leva di comando
        MOV     DX,0          ;DX=0 per informazione pulsanti
        INT     15H
        MOV     CL,04         ;rotazione di quattro bit
        ROR     AL,CL         ;bit 7-4 nei bit 3-0
        AND     AX,0FH        ;isola i quattro bit meno significativi
        JMP     SEND          ;restituzione dell'informazione

;codice per il campionamento di quattro potenziometri (A(X), A(Y), B(X) e B(Y))
POT:    MOV     AH,84H        ;interfaccia leva comando
        MOV     DX,01H        ;lettura potenziometri
        INT     15H

;codice per restituire i valori dei registri AX, BX, CX e DX.
SEND:   MOV     DI,[BP]+12    ;invio valore del potenziometro A(X)
        MOV     [DI],AX
        MOV     DI,[BP]+10    ;invio valore del potenziometro A(X)
        MOV     [DI],BX
        MOV     DI,[BP]+8     ;invio valore del potenziometro A(X)
        MOV     [DI],CX
        MOV     DI,[BP]+6     ;invio valore del potenziometro A(X)
        MOV     [DI],DX

        MOV     SP,BP         ;parametri di ritorno
        POP     BP
        RET     10

GIOCOB  ENDP                  ;fine della procedura GIOCOB
CODICE  ENDS                  ;fine del segmento di codice

        END                   ;fine del programma
```

Figura 10.11 Codice in linguaggio assemblatore per interfacciare il joystick con il programma principale scritto in Pascal

scal. La procedura GIOCOP deve essere dichiarata PUBLIC per rendere possibile il passaggio dei parametri con il programma principale e deve essere di tipo FAR, per cui necessita – al termine della sua codifica – di una istruzione di ritorno RET di tipo FAR. Il contenuto di BP (Base Pointer) viene salvato sullo stack, in modo che il valore del registro SP (Stack Pointer) possa essere trasferito al registro BP. Generalmente, il passaggio dei parametri avviene tramite lo stack. Le variabili possono essere trasferite dal programma principale al programma scritto in linguaggio assemblatore attraverso il registro SI (Indice Sorgente) e, viceversa, attraverso il registro DI (Indice Destinazione). In questo esempio, cinque variabili di tipo word (DW) vengono trasferite tra i due programmi. Poiché GIOCOP è una procedura di tipo FAR, l'indirizzo di ritorno occupa i primi otto byte dello stack. L'indirizzo della variabile del programma principale (BY) si trova in [BP]+6 e viene trasferito nel registro DI. Il valore intero contenuto nel registro DX viene poi trasferito in quella locazione di memoria con l'istruzione MOV [DI],DX. Avanzando di altri due byte sullo stack, si trova l'indirizzo della variabile successiva (BX), e così via. In questo modo, vengono restituiti al programma principale quattro valori interi, che rappresentano le letture del potenziometro del joystick, supponendo che inizialmente il registro DX sia stato inizializzato a 1. Se DX invece contiene il valore 0, solo il valore restituito al registro AX ha significato e corrisponde all'informazione relativa

```
PROGRAM GIOCOPMS(INPUT,OUTPUT);

VAR
  OPER,AX,AY,BX,BY:INTEGER;
  CH:CHAR;

PROCEDURE GIOCOP(VAR OPER,AX,AY,BX,BY:INTEGER);
  EXETRNAL;

PROCEDURE INFORM;
BEGIN
  OPER:=1;
  GIOCOP(OPER,AX,AY,BX,BY);
  WRITELN(AX,'  ',AY,'  ',BX,'  ',BY)
END;

BEGIN {PROCEDURA PRINCIPALE}
  REPEAT
    INFORM;
    UNTIL CH='Q'
END.
```

Figura 10.12 Programma in Pascal che chiama il programma in linguaggio assemblatore per campionare la porta del joystick

ai pulsanti. Utilizzando la stessa tecnica, è possibile realizzare un passaggio illimitato di parametri.
Prima di abbandonare questa routine, viene ripristinato il contenuto del registro BP e viene eseguita l'istruzione di ritorno RET di tipo FAR. L'operando 10 dell'istruzione RET indica che lo stack deve essere svuotato dei parametri in esso contenuti (10 byte di memoria), che erano stati passati alla routine dal programma chiamante.
La Figura 10.12 mostra il programma principale in Pascal. Si tenga presente che il programma in linguaggio assemblatore viene invocato utilizzando l'istruzione di chiamata ad una procedura esterna. Questo programma si chiama GIOCOP.ASM e, insieme alla versione in codice oggetto .OBJ, è contenuto nel drive B. Un'altra considerazione da fare riguarda l'ordine con cui le variabili vengono passate. OPER, AX, AY, BX e BY sono variabili intere: OPER passa al programma in linguaggio assemblatore (registro DX) il valore 1 o 0 e il suo indirizzo è [BP]+14, mentre AX, AY, BX e BY costituiscono i parametri di ritorno al programma principale. Si tenga presente che l'indirizzo di BY è [BP]+6.
I passi necessari per creare l'esempio di interfaccia con il programma Pascal sono abbastanza semplici. Innanzitutto, viene scritto il programma in linguaggio assemblatore con un editor di video (ad esempio l'editor di Norton). Poi, il programma viene tradotto in codice macchina, nel modo seguente:

```
B>C:MASM GIOCOP;
IBM Personal Computer MACRO Assembler  Version 2.00
(C)Copyright IBM Corp 1981, 1984
(C)Copyright Microsoft Corp 1981, 1983, 1984

50096 Bytes free

Warning Severe
Errors  Errors
0       0
```

In questa fase viene creato il file .OBJ sul dischetto che risiede nel drive B. Successivamente, viene scritto il programma in Pascal con lo stesso editor e viene compilato con il compilatore Pascal della IBM per creare un file di codice oggetto .OBJ, utilizzando il seguente comando:

```
A>PAS1 B:GIOCOPMS,B:,,;

IBM Personal Computer Pascal Compiler
Versione 2.00
(C)Copyright IBM Corp 1981, 1984
(C)Copyright Microsoft Corp 1981, 1984

 Pass One    No Errors Detected

A>PAS2

 Code Area Size = #00DB   (  219)
```

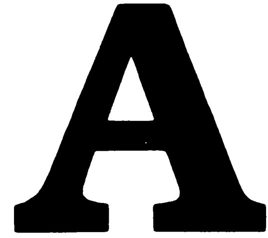
```
Cons Area Size = #000D   (    13)
Data Area Size = #0024   (    36)

Pass Two    No Errors Detected
```

Da ultimo, i due file .OBJ devono essere collegati insieme, utilizzando il linker di sistema (per Linker Personal Computer IBM, versione 2.30):

```
A>LINK B:GIOCOPMS+B:GIOCOP,B:,,A:;
```

In questo esempio, la versione compilata del programma principale in Pascal (GIOCOPMS) risiede nel dischetto presente nel drive B, dove risiede anche la versione in codice oggetto GIOCOP.OBJ del programma scritto in linguaggio assemblatore. Questi programmi vengono collegati insieme utilizzando l'opzione (+) del linker. Conclusa questa operazione, non è più necessario mantenere su dischetto il programma in linguaggio assemblatore. Il programma principale GIOCOPMS.EXE costituisce la versione eseguibile finale.



Il Macro Assembler IBM

Questa appendice costituisce una guida all'utilizzo del Macro Assembler IBM. Viene presentato, in particolare, un esempio che illustra come scrivere e tradurre in codice macchina un programma in linguaggio assemblatore e come risolvere i riferimenti esterni in esso presenti (vedere Figura A.1).
A partire dal codice sorgente scritto con l'editor, viene spiegato come utilizzare l'assembler per generare il codice oggetto (file di tipo .OBJ), che significato hanno i file .LST e di cross-reference e perché sono necessari. Dopo la creazione del codice oggetto, viene illustrato come ottenere la versione eseguibile (file di tipo .EXE) tramite il linker e, in particolare, viene discusso il significato e l'utilità del file di tipo .MAP.
Inoltre, questa appendice elenca le differenze esistenti tra un file .EXE e un file .COM e insegna come utilizzare il programma EXE2BIN.EXE (fornito con il DOS) per convertire un file di tipo .EXE in un file di tipo .COM.

A.1 Informazioni di carattere generale

Esistono due versioni di Macro Assembler IBM: ridotta e completa. Sebbene la versione ridotta richieda meno memoria, non supporta tutte le funzioni e le opzioni offerte dalla versione completa. Per questa ragione, concentreremo la nostra discussione sulla versione completa (che chiameremo macro assembler).
Il macro assembler opera con almeno 128 kB di memoria (a partire dalla versione DOS 2.0) e durante la traduzione in codice macchina utilizza fino a 192 kB di memoria.

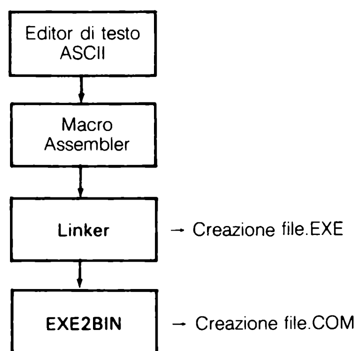


Figura A.1 Passi per la creazione di un programma in linguaggio assemblatore

A.2 Creazione del codice sorgente in linguaggio assemblatore

Il ciclo di sviluppo di applicazioni in linguaggio assemblatore ha inizio con la definizione del problema che deve essere risolto a livello macchina. Ad esempio, può essere richiesta la lettura dei valori resistivi del joystick collegato al calcolatore.
Prima di pensare alla soluzione programmativa, è indispensabile capire come il calcolatore acceda alle informazioni riguardanti il joystick (consultate a questo proposito le funzioni BIOS elencate nei manuali tecnici IBM).
Come abbiamo detto nel Capitolo 2, il programmatore utilizza la codifica mnemonica per scrivere il programma sorgente, nel rispetto delle regole grammaticali (sintassi) definite dall'assembler affinché la traduzione in codice oggetto proceda correttamente. Il codice sorgente può essere scritto con un qualsiasi editor di video (ad esempio il Norton Editor, l'IBM Professional Editor o il WordStar), che genera un file di uscita nel formato ASCII.
Nel nostro esempio, scriviamo il seguente codice sorgente nel file GIOCO.ASM. L'estensione .ASM indica che si tratta di un file contenente codice sorgente nel linguaggio assemblatore.

```
;per macchine 80286/80386
;programma che campiona il joystick e restituisce i valori di
;potenziometro nei registri indicati.

PAGE ,132                           ;dimensionamento pagina

CODICE    SEGMENT PARA 'CODICE'     ;definisce il segmento di codice
PROCEDURA PROC    FAR               ;inizio della procedura
```

```
          ASSUME  CS:CODICE
          PUSH    DS              ;salva DS sullo stack
          SUB     AX,AX           ;azzera AX
          PUSH    AX              ;salva 0 sullo stack

;campionamento potenziali del joystick. AX=A(X), ;CX=B(X),
DX=B(Y) dopo l'interruzione
          MOV     AH,84H          ;campionamento leva di comando
          MOV     DX,01H          ;campionamento potenziali
          INT     15H

          RET                     ;il controllo ritorna al DOS
PROCEDURA ENDP                    ;fine della procedura
CODICE    ENDS                    ;fine del segmento di codice

          END                     ;fine del programma
```

Dopo aver scritto il codice sorgente, è possibile passare alla seconda fase del ciclo di sviluppo dell'applicazione, cioè all'utilizzo del macro assembler per tradurre il codice sorgente (scritto in una codifica mnemonica ispirata alla lingua inglese) in una forma che sia comprensibile al microprocessore.

A.3 Uso del Macro Assembler

Il Macro Assembler IBM esamina il codice sorgente – cioè il contenuto del file .ASM – ed esegue la traduzione in codice oggetto, cioè genera il file .OBJ. Questa operazione necessita di due passate da parte dell'assembler. La prima passata definisce l'offset di rilocazione per ogni linea di codice, costruisce una tabella dei simboli, ma non genera codice oggetto (consultare l'opzione /D nella Tabella A.1); la seconda passata utilizza i risultati prodotti dalla prima passata per generare codice oggetto.

ERRORI DI FASE

La seconda passata può generare errori di fase. Un errore di fase indica che l'assembler ha scoperto, durante la seconda passata, che una variabile, una etichetta o una procedura ha un indirizzo differente rispetto a quello che era stato riportato nella tabella dei simboli durante la prima passata. Un errore di questo tipo può essere dipeso da alterazioni non coordinate di costanti mnemoniche, o da un uso di istruzioni che contrastano con le regole definite dall'assembler.
Supponiamo che il programma sorgente GIOCO.ASM si trovi nel drive B, che il Macro Assembler IBM sia nel drive A e che il drive B sia quello selezionato. Compare allora sullo schermo il prompt B>.

Tabella A.1 Parametri per il Macro Assembler IBM

Opzione	Descrizione
/A	Questa opzione (attiva per default) indica all'assembler di elencare i segmenti del codice sorgente in ordine alfabetico.
/D	Quando si verifica un errore di fase, è stata riscontrata una incompatibilità tra i valori generati dalla prima passata e quelli generati dalla seconda passata nel processo di traduzione. Per localizzare questa incongruenza, si può informare l'assembler di generare un listato durante la prima passata, utilizzando l'opzione /D. Questo listato addizionale può essere confrontato con quello che viene generato automaticamente dalla seconda passata.
/E	Questa opzione indica all'assembler di tradurre il codice sorgente per l'80287/80387 e di generare costanti in virgola mobile nella forma attesa dal coprocessore. Opera nello stesso modo dell'opzione /R.
/N	Questa opzione informa l'assembler di non generare una tabella di simboli al termine del file.LST.
/O	Questa opzione indica all'assembler di generare la codifica ottale equivalente della traduzione in linguaggio macchina.
/R	Avviene la traduzione del codice sorgente in una forma compatibile con il coprocessore 80287/80387 per quanto riguarda i valori numerici prodotti. Il codice macchina generato con questa opzione non può essere eseguito correttamente su sistemi che non dispongono del coprocessore.
/S	Questa opzione disattiva l'opzione di default /A, per cui i segmenti vengono elencati nello stesso ordine con cui appaiono nel codice sorgente.

Per invocare l'assembler è sufficiente scrivere:

```
B>A:MASM
```

Sullo schermo vengono visualizzate le seguenti linee di testo:

```
IBM Personal Computer MACRO Assembler   Version 2.00
(C)Copyright IBM Corp 1981, 1984
(C)Copyright Microsoft Corp 1981, 1983, 1984

Source filename [.ASM]
```

L'assembler richiede all'utente di specificare il nome del file sorgente che deve essere tradotto in codice macchina. Le parentesi quadre [ ] contengono l'estensione per default (.ASM) che viene assegnata al file sorgente dall'assembler. Poiché il codice sorgente GIOCO è stato salvato con l'estensione .ASM, scrivete da tastiera solo GIOCO e premete il tasto ENTER come risposta alla prima richiesta del calcolatore.
Si tenga presente che la versione 2.00 del Macro Assembler IBM permette

di specificare le indicazioni del drive e del sottodirettorio (path name) se state utilizzando dischi fissi.
Dopo aver premuto il tasto ENTER, viene visualizzata una seconda richiesta:

Object filename [GIOCO.OBJ]:

L'assembler indica che, a meno di esplicita controindicazione, il nome del codice oggetto sarà GIOCO.OBJ. Premete il tasto ENTER se accettate questo identificatore per default e vedrete visualizzata sullo schermo una terza richiesta:

Source Listing [NUL.LST]:

Un file con l'estensione .LST (listato) costituisce il prodotto stampabile delle operazioni dell'assembler poiché contiene i numeri di linea, la traduzione equivalente in codice macchina di ogni istruzione e una tabella di simboli. L'assembler non genera questo file per default, come viene indicato dal mnemonico NUL.
Perché l'assembler produca il file .LST, dovete scrivere da tastiera GIOCO e premere il tasto ENTER. Non è necessario specificare da tastiera l'estensione .LST, in quanto questa viene assunta per default dall'assembler.
Dopo aver premuto il tasto ENTER, l'assembler visualizza sullo schermo la quarta e ultima richiesta:

Cross reference [NUL.CRF]:

L'assembler genera il file .CRF che verrà impiegato dall'utilità CREF.EXE per creare il listato finale di cross-reference (nel quale, accanto a ogni simbolo usato, compare l'elenco dei numeri di linea in cui tale simbolo è citato). Ancora una volta l'assembler non crea per default il file .CRF (mnemonico NUL), per cui, per ottenerlo, dovete specificare da tastiera il nome GIOCO e premere il tasto ENTER. L'assembler allora crea il file GIOCO.CRF, che verrà utilizzato in seguito.
L'assembler ora esamina il file sorgente GIOCO.ASM e produce i file .OBJ, .LST e .CRF. Per assicurarsi che queste operazioni siano state eseguite correttamente, richiedete su video il direttorio del disco B:

```
GIOCO   ASM    758   12-27-85   6:47p
GIOCO   OBJ     57   12-29-85   2:55p
GIOCO   LST   1656   12-29-85   2:55p
GIOCO   CRF     69   12-29-85   2:55p
```

È noto che il file GIOCO.ASM contiene il codice ASCII del programma e che il file GIOCO.OBJ non può essere visualizzato, in quanto rappresenta la versione in codice macchina del programma. Anche il file GIOCO.CRF non può essere visualizzato, in quanto costituisce un passo intermedio nella genera-

zione di una tabella di cross-reference. Il file che può invece essere visualizzato è GIOCO.LST.
Come abbiamo detto precedentemente, il file .LST contiene la versione tradotta del codice sorgente, l'indicazione dei numeri di linea, la rappresentazione equivalente in codice macchina delle istruzioni e una tabella di simboli. Questo file di solito viene consultato in fase di correzione e di analisi del programma e può essere stampato inviando da tastiera i comandi TYPE GIOCO.LST, CTRL-PRTSC e ENTER:

```
IBM Personal Computer MACRO Assembler   Version 2.00    Page 1-1
                                                   12-29-85

1              ;per macchine 80286/80386
2              ;programma che campiona il joystick e restituisce i valori di
3              ;potenziometro nei registri indicati.
4
5
6              PAGE ,132                          ;dimensionamento pagina
7
8    0000      CODICE    SEGMENT PARA 'CODICE'    ;definisce il segmento di codice
9    0000      PROCEDURA PROC    FAR              ;inizio della procedura
10                       ASSUME  CS:CODICE
11   0000 1E             PUSH    DS               ;salva DS sullo stack
12   0001 2B C0          SUB     AX,AX            ;azzera AX
13   0003 50             PUSH    AX               ;salva 0 sullo stack
14
15             ;campionamento potenziali del joystick. AX=A(X), CX=B(X), DX=B(Y)
16             ;dopo l'interruzione
17   0004 B4 84          MOV     AH,84H           ;campionamento leva di comando
18   0006 BA 0001        MOV     DX,01H           ;campionamento potenziali
19   0009 CD 15          INT     15H
20
21   000B CB             RET                      ;il controllo ritorna al DOS
22   000C      PROCEDURA ENDP                     ;fine della procedura
23   000C      CODICE    ENDS                     ;fine del segmento di codice
24
25                       END                      ;fine del programma

IBM Personal Computer MACRO Assembler   Version 2.00    Page   Symbols-1
                                                   12-29-85

Segment and Groups:

          Name                Size    Align   Combine    Class

CODICE....................    000C    PARA    NONE       'CODICE'

Symbols:

          Name                Type    Value   Attr

PROCEDURA.................    F PROC  0000    CODICE     Length =000C

50096 Bytes free

Warning Severe
Errors  Errors
0       0
```

OPZIONI

Il Macro Assembler IBM offre molte opzioni all'utente per tradurre un programma. Le opzioni vengono specificate dopo il comando e iniziano con una barra (/). Sono elencate in ordine alfabetico in Tabella A.1 (Per maggiori dettagli, consultate l'*IBM Macro Assembler Reference Manual*).

COMANDI IN FORMA ABBREVIATA

È possibile invocare l'assembler con comandi in forma abbreviata, che risultano particolarmente utili ai programmatori esperti.
Come abbiamo detto al Capitolo 2, quando viene scritto da tastiera:

```
B>A:MASM GIOCO
```

l'assembler immediatamente genera il file .OBJ, senza richiedere all'utente di specificare il nome che egli intende assegnare al file di codice oggetto. Inoltre, con il precedente comando, l'assembler non genera i file .LST e .CRF.
I seguenti comandi in forma abbreviata evitano al programmatore di dover rispondere alle richieste dell'assembler, in quanto vengono assegnati ai file .LST e .CRF gli identificatori per default.

```
B>A:MASM GIOCO,,,
```

Ogni virgola (,) disattiva una richiesta dell'assembler e attiva automaticamente per l'identificatore di file la codifica per default. Il comando NUL svolge la stessa funzione della virgola.
Le opzioni vengono specificate nel modo seguente:

```
B>A:MASM GIOCO/N
```

Questo comando informa l'assembler di non generare una tabella di simboli al termine del file .LST.

A.4 Cross-reference mediante CREF.EXE

Le informazioni di cross-reference sono molto utili in fase di correzione dei programmi, in quanto forniscono al programmatore un elenco in ordine alfabetico di dati, variabili, etichette, costanti e altri simboli che sono stati referenziati nel codice sorgente codice. Il listato include il numero di linea in corrispondenza del quale il simbolo è definito e l'indicazione di tutte le altre linee in cui il simbolo stesso è stato utilizzato.
Per generare un listato di cross-reference si deve selezionare la quinta op-

zione dell'assembler, che permette la generazione del file intermedio *nomefile*.CRF. Questo file viene utilizzato dal programma di utilità CREF.EXE per creare l'effettivo file di cross-reference.
Per generare il listato di cross-reference, scrivete semplicemente da tastiera:

```
B>CREF
```

e premete il tasto ENTER. Sullo schermo appare allora:

```
The IBM Personal Computer CREF,  Version 2.00
(C)Copyright IBM Corp 1981, 1984
(C)Copyright Microsoft Corp 1981, 1983, 1984

Cref filename [.CREF]
```

L'utilità CREF.EXE richiede il nome del file *nomefile*.CRF. Si risponda scrivendo da tastiera GIOCO e premendo il tasto ENTER. Si ricordi che non è indispensabile aggiungere l'estensione di file .CRF, poiché questa viene assegnata per default.
Dopo aver eseguito l'operazione, viene visualizzata sullo schermo la seguente richiesta:

```
List filename [GIOCO.REF]:
```

Il nome del file per default, che viene assegnato da CREF.EXE, è GIOCO.REF. Questo nome può essere modificato dall'utente, specificando – se necessario – un differente indicatore di drive. Se, invece, si accetta questo identificatore per default, è sufficiente premere il tasto ENTER.

CHIAMATA DI CREF.EXE IN FORMA ABBREVIATA

Come per il macro assembler, esiste un comando in forma ridotta per invocare l'utilità di cross-reference. Scrivendo da tastiera:

```
B>CREF GIOCO
```

si impone al programma CREF.EXE di cercare automaticamente il file GIOCO.CRF e poi di generare un file .REF con lo stesso nome. Nel nostro esempio, quindi, questo file si chiama GIOCO.REF.
Se questa operazione ha luogo senza errori, sullo schermo viene visualizzata l'indicazione del drive selezionato (prompt). Per assicurarsi che il file sia stato effettivamente creato, è sufficiente richiedere il direttorio del dischetto B: nel nostro caso, deve comparire il file GIOCO.REF. Se scrivete da tastiera TYPE GIOCO.REF, viene visualizzato sullo schermo il listato di

cross-reference:

```
B>TYPE GIOCO.REF


Symbol Cross Reference                    (# is definition)
Cref-1

CODICE . . . . . . . . . . . . . . 8

CODICE . . . . . . . . . . . . . . 8#   10   23
PROCEDURA. . . . . . . . . . . . . 9#   22

3 Symbols

63048 Bytes Free
```

Da questo listato è possibile determinare il numero di segmenti che sono stati definiti nel programma. Nel nostro caso, esiste un solo segmento: CODICE. Si possono anche notare le dichiarazioni delle entità simboliche CODICE e PROCEDURA. Il carattere #, che segue un numero di linea, indica che il relativo simbolo (ad esempio CODICE o PROCEDURA) è stato definito su quella linea. I numeri di linea seguenti specificano invece le linee in cui il simbolo è stato referenziato.
Anche in questo breve esempio, è possibile utilizzare il listato di cross-reference per assicurarsi che ogni sezione di codice sia stata conclusa con la direttiva END, come nel caso di CODICE (linea 23) e PROCEDURA (linea 22). Aumentando la dimensione del programma scritto in linguaggio assemblatore, cresce l'importanza delle informazioni di cross-reference, per garantire una rapida correzione del codice sorgente.
Si può ottenere una copia di questo file semplicemente accendendo la stampante e inviando da tastiera il comando TYPE seguito dal nome del file, come viene qui di seguito indicato:

```
B>TYPE GIOCO.REF
```

Si prema CTRL-PRTSC e poi il tasto ENTER. Premete nuovamente CTRL-PRTSC per terminare la stampa.
Sia che abbiate generato il listato di cross-reference, sia che non lo abbiate generato, il passo successivo consiste nel risolvere i riferimenti esterni presenti nel file .OBJ. Questa operazione viene eseguita invocando il programma di utilità LINK.EXE che viene fornito insieme all'assembler.

A.5 Collegamento: LINK.EXE

Il linker (LINK.EXE) esamina il file .OBJ e genera un programma rilocabile. In questo modo, il sistema operativo viene abilitato a caricare in una qua-

lunque area di memoria disponibile la versione eseguibile del programma scritto in linguaggio assemblatore. L'alternativa alla coesistenza di più programmi in memoria è di definire per il programma indirizzi fisici prestabiliti. Supponiamo che il programma inizi dalla locazione di memoria 0100H e che il risultato delle elaborazioni sia stato memorizzato nella locazione 4F3AH. Se un altro programma referenzia la stessa locazione, si crea un problema di condivisione di memoria. Un programma rilocabile, invece, può essere allocato in memoria in corrispondenza di locazioni libere perché tutti gli indirizzi si modificano automaticamente in fase di esecuzione, grazie alle modifiche apportate dal linker al codice oggetto.
Il linker esegue altre operazioni estremamente utili, come ad esempio quella di collegare file di tipo .OBJ, che sono stati compilati separatamente. (È possibile così sezionare in moduli più piccoli un programma in linguaggio assemblatore di grandi dimensioni). Il linker può anche risolvere i riferimenti tra il programma principale e le routine chiamate appartenenti a librerie esterne.
Per invocare il linker, inviate da tastiera il comando LINK. Saranno allora visualizzate le seguenti linee di testo:

```
IBM Personal Computer Linker

Version 2.20 (C)Copyright IBM Corp 1981, 1982, 1983, 1984

Object Modules [.OBJ]:
```

Il linker richiede all'utente il nome del file .OBJ da collegare.
Nel nostro esempio, si scriva da tastiera GIOCO. Ancora una volta, non è necessario specificare l'estensione di file .OBJ, in quanto il linker la assume per default. Viene allora visualizzata la seguente richiesta:

```
Run file [GIOCO.EXE]:
```

Per default, il linker fornisce il nome della versione eseguibile del programma. È possibile sostituire tale nome con un altro, senza dover specificare l'estensione .EXE.
Si prema il tasto ENTER se, invece, intendete accettare questo identificatore di file. La successiva richiesta visualizzata sullo schermo risulta:

```
List File [NUL.MAP]:
```

Il file .MAP contiene l'elenco dei segmenti che sono stati definiti nel codice sorgente e specifica i valori di offset presenti nel file eseguibile, per cui si rivela molto utile in fase di correzione del programma.
Il linker non genera questo file per default (come viene indicato dal mnemonico NUL contenuto tra le parentesi quadre) ma, per ottenerlo, dovete speci-

ficare il nome GIOCO e poi premere il tasto ENTER. Da ultimo, viene visualizzata la quarta richiesta:

```
Libraries [.LIB]:
```

Il linker chiede ora il nome delle librerie contenenti le routine che sono referenziate nel codice sorgente. Nel nostro esempio, non è stata utilizzata alcuna libreria definita dall'utente, per cui è sufficiente premere il tasto ENTER per andare avanti.
A questo punto appaiono i seguenti messaggi:

```
Warning: No STACK segment
There was 1 error detected.
```

Questi messaggi indicano che il codice sorgente del nostro programma non contiene la definizione di un segmento di stack. In questo caso, il messaggio può essere ignorato in riferimento al tipo di file che si intende creare. Il programma di esempio GIOCO.ASM è stato scritto per essere convertito in un file estremamente compatto chiamato file .COM.
Richiediamo ora il direttorio del disco B, per assicurarci che ogni passo del ciclo di sviluppo dell'applicazione sia stato compiuto correttamente:

```
GIOCO   ASM    758   12-27-85   6:47p
GIOCO   OBJ     57    1-02-86   6:36p
GIOCO   LST   1656    1-02-86   6:36p
GIOCO   CRF     69    1-02-86   6:36p
GIOCO   REF    268    1-02-86   6:36p
GIOCO   MAP    154    1-02-86   7:13p
GIOCO   EXE    524    1-02-86   7:13p
        7 File(s)   4159488 bytes free
```

A questo punto potete lanciare il file GIOCO.EXE che costituisce la versione rilocabile, eseguibile del programma GIOCO.
Prima di convertire questo file in un file .COM, discutiamo alcune opzioni del linker e indichiamo il significato di alcuni comandi espressi in forma abbreviata.

OPZIONI DEL LINKER E COMANDI IN FORMA ABBREVIATA

Le opzioni elencate in Tabella A.2 possono essere incluse nel comando di invocazione del linker, facendole precedere dal carattere barra (/).
Come per il macro assembler, esistono alcune forme abbreviate di comandi che possono essere utilizzate per invocare il Linker e che vengono elencate in Tabella A.3.

Tabella A.2 Opzioni del linker

Opzione	Descrizione
/DS ALLOCATION	Questa opzione (abbreviazione /DS) informa il linker di caricare tutti i dati definiti nell'area dati in corrispondenza dell'estremità superiore di quell'area. Per default, il dato viene caricato in corrispondenza dell'estremità inferiore dell'area, iniziando dall'offset 0.
/HIGH	Questa opzione (abbreviazione /H) informa il linker di caricare il file eseguibile nella zona di memoria superiore, senza interferire con il codice COMMAND.COM.
/LINE	/L informa il linker di includere gli indirizzi e i numeri di linea di ognuna delle istruzioni sorgenti dei moduli di ingresso nel file.LST.
/NODEFAULT LIBRARY SEARCH	Questa opzione (abbreviazione /NOD) informa il linker di non richiedere all'utente di specificare le librerie per default.
/NOGROUP ASSOCIATION	Questa opzione (abbreviazione /NOG) informa il linker di correggere gli indirizzi esterni di tipo long, anche se il simbolo è stato definito in un segmento interno ad un'area definita.
/PAUSE	/P informa il linker di generare su video un messaggio che richieda l'inserimento nel drive di un dischetto per contenere il file eseguibile.
/STACK:*size*	/S seguita da un valore decimale compreso tra 0 e 65 536 permette al programmatore di specificare la dimensione del segmento di stack. I valori compresi tra 0 e 512 assegnano una quantità minima di stack di 512 byte.

Tabella A.3 Comandi in forma abbreviata per il linker

Comando	Descrizione
LINK *nomefile*;	Questo comando in forma abbreviata invoca il linker, che utilizza il file *nomefile*.OBJ in ingresso e automaticamente identifica il file di uscita con *nomefile*.EXE.
LINK *nomefile*,,;	Questo comando in forma abbreviata indica al linker di cercare il file *nomefile*.OBJ in ingresso e di generare i file di uscita *nomefile*.EXE e *nomefile*.MAP.

Nota: Ogni virgola (,) disattiva una richiesta del linker e attiva il nome per default del file corrispondente a quella particolare opzione. Il nome NUL può essere sostituito alla virgola per disattivare una particolare opzione.

CONFRONTO TRA FILE .EXE E FILE .COM

I vantaggi di un file .EXE sono due: innanzitutto, il file è rilocabile, per cui più di un programma può essere caricato in memoria sfruttando lo spazio disponibile; secondariamente, i file di tipo.EXE permettono di utilizzare fino a quattro segmenti: STACK, DATI, CODICE e EXTRA. In questo modo, è possibile raggiungere una buona modularità del codice e creare programmi di grandi dimensioni. Lo svantaggio di un file .EXE è costituito dalla quantità di codice che il linker aggiunge per rendere possibile la rilocazione. Nel nostro esempio, la dimensione del file contenente il programma GIOCO.EXE è 524 byte.
Al contrario, un file .COM può contenere un solo segmento, in cui risiedono le informazioni relative allo STACK, ai DATI e al CODICE. Quando la dimensione di un programma in linguaggio assemblatore non è elevata, un file di tipo .COM risulta più che sufficiente per includere tutti i dati, le istruzioni e le operazioni di gestione dello stack. Diversamente dal file .EXE, il file .COM non è rilocabile e deve iniziare all'indirizzo 0100H. Il vantaggio di un file .COM rispetto ad un file .EXE è che il file .COM riduce drasticamente la dimensione del file eseguibile. Per creare un file .COM, viene fornito assieme al DOS il programma di utilità EXE2BIN.EXE.

A.6 Creazione dei file .COM

Il programma di utilità EXE2BIN.EXE genera la versione .COM da un file .EXE. Per chiamarlo, scrivete da tastiera:

```
B>A:EXE2BIN GIOCO GIOCO.COM
```

e poi premete il tasto ENTER. Supponiamo che il programma EXE2BIN.EXE si trovi sul dischetto inserito nel drive A, per cui basta scrivere A:EXE2BIN per referenziare il programma. Il file di tipo .EXE da convertire è GIOCO (si noti che non è indispensabile specificare l'estensione .EXE), mentre il nome del file di tipo .COM da creare è GIOCO.COM. Si ricordi, in questo caso, di specificare l'estensione di file, perché altrimenti il il programma EXE2BIN adotta l'estensione per default .BIN, che non rappresenta un'estensione corretta per un file eseguibile (cambiatela eventualmente con il comando RENAME).
Confrontiamo ora la dimensione in byte dei file GIOCO.EXE e GIOCO.COM:

```
GIOCO   EXE    524   1-02-86   7:45p
GIOCO   COM     12   1-02-86   7:45p
```

Si noti l'enorme riduzione di codice ottenibile con la versione di tipo .COM.

La Tabella A.4 contiene un elenco dei programmi di utilità che vengono forniti con il Macro Assembler IBM, oltre all'indicazione dei file che tali programmi ricevono in ingresso e dei file che producono in uscita.

Tabella A.4 Specifiche di file di ingresso/uscita per le utilità che vengono fornite insieme al Macro Assembler IBM

Programma di utilità	**Descrizione**
MASM	
File di ingresso:	*nomefile*.ASM (file di testo ASCII)
File di uscita:	*nomefile*.OBJ (codice linguaggio macchina)
	nomefile.LST (listato)
	nomefile.CRF (file simbolico usato da CREF)
CREF	
File di ingresso:	*nomefile*.CRF (file simbolico usato da CREF)
File di uscita:	*nomefile*.REF (file di cross-reference)
LINK	
File di ingresso:	*nomefile*.OBJ (codice linguaggio macchina)
	nomelib.LIB (libreria definita dall'utente)
File di uscita:	*nomefile*.EXE (file eseguibile)
	nomefile.MAP (file di mappa della memoria)
EXE2BIN	
File di ingresso:	*nomefile*.EXE (codice ad un segmento)
File di uscita:	*nomefile*.COM (file eseguibile)

B

Il Macro Assembler Microsoft

Questa appendice costituisce una guida all'utilizzo del Macro Assembler Microsoft. Viene presentato, in particolare, un esempio che illustra come scrivere e tradurre in codice macchina un programma in linguaggio assemblatore e come risolvere i riferimenti esterni in esso presenti.
A partire dal codice sorgente scritto con l'editor, viene spiegato come utilizzare l'assembler per generare il codice oggetto (file di tipo .OBJ), che significato hanno i file .LST e di cross-reference e perché sono necessari. Dopo la creazione del codice oggetto, viene illustrato come ottenere la versione eseguibile (file di tipo .EXE) tramite il linker e, in particolare, viene discusso il significato e l'utilità del file di tipo .MAP.
Inoltre, questa appendice elenca anche le differenze esistenti tra un file .EXE e un file .COM e insegna come utilizzare il programma EXE2BIN.EXE (fornito con il DOS) per convertire un file di tipo .EXE in un file di tipo .COM.

B.1 Informazioni di carattere generale

Il Macro Assembler Microsoft è in grado di tradurre in codice macchina i programmi che vengono eseguiti sui microprocessori 8086, 80186 e 80286 e sui coprocessori 8087 e 80287. Questo assembler richiede la presenza nel sistema della versione 2.0 del DOS, o delle versioni successive, e un minimo di 128 kB di memoria.

B.2 Creazione del codice sorgente in linguaggio assemblatore

Il ciclo di sviluppo di applicazioni in linguaggio assemblatore ha inizio con la definizione del problema che deve essere risolto a livello macchina. Ad esempio, può essere richiesta la lettura dei valori resistivi del joystick collegato al calcolatore.
Prima di pensare alla soluzione programmativa, è indispensabile capire come il calcolatore acceda alle informazioni riguardanti il joystick (consultate a questo proposito le funzioni BIOS elencate nell'*IBM Technical Reference Manual*).
Come abbiamo detto nel Capitolo 2, il programmatore utilizza la codifica mnemonica per scrivere il programma sorgente, nel rispetto delle regole grammaticali (sintassi) definite dall'assembler affinché la traduzione in codice oggetto proceda correttamente. Il codice sorgente può essere scritto con un qualsiasi editor di video (ad esempio il Norton Editor o l'IBM Professional Editor o WordStar), che genera un file di uscita nel formato ASCII.
Nel nostro esempio, scriviamo il seguente codice sorgente nel file GIOCO.ASM. L'estensione .ASM indica che si tratta di un file contenente codice sorgente nel linguaggio assemblatore.

```
;per macchine 80286/80386
;programma che campiona il joystick e restituisce i valori di
;potenziometro nei registri indicati.

PAGE ,132                              ;dimensionamento pagina

CODICE    SEGMENT PARA 'CODICE'   ;definisce il segmento di codice
PROCEDURA PROC    FAR                  ;inizio della procedura
          ASSUME  CS:CODICE
          PUSH    DS                   ;salva DS sullo stack
          SUB     AX,AX                ;azzera AX
          PUSH    AX                   ;salva 0 sullo stack

;campionamento potenziali del joystick. AX=A(X), CX=B(X), DX=B(Y)
;dopo l'interruzione
          MOV     AH,84H               ;campionamento leva di comando
          MOV     DX,01H               ;campionamento potenziali
          INT     15H

          RET                          ;il controllo ritorna al DOS
PROCEDURA ENDP                         ;fine della procedura
CODICE    ENDS                         ;fine del segmento di codice

          END                          ;fine del programma
```

Dopo aver scritto il codice sorgente, è possibile passare alla seconda fase del ciclo di sviluppo dell'applicazione, cioè all'utilizzo del macro assembler per tradurre il codice sorgente (scritto in una codifica mnemonica ispirata alla lingua inglese) in una forma che sia comprensibile al microprocessore.

B.3 Uso del Macro Assembler

Il Macro Assembler Microsoft esamina il codice sorgente – cioè il contenuto del file .ASM – ed esegue la traduzione in codice oggetto, cioè genera il file .OBJ. Questa operazione necessita di due passate da parte dell'assembler. La prima passata definisce l'offset di rilocazione per ogni linea di codice e costruisce una tabella dei simboli, ma non genera codice oggetto (consultare l'opzione /D nella Tabella B.1); la seconda passata utilizza i risultati prodotti dalla prima passata per generare codice oggetto.

ERRORI DI FASE

La seconda passata può generare errori di fase. Un errore di fase indica che l'assembler ha scoperto, durante la seconda passata, che una variabile, una etichetta o una procedura ha un indirizzo differente rispetto a quello che era stato riportato nella tabella dei simboli durante la prima passata. Un errore di questo tipo può essere dipeso da alterazioni non coordinate di costanti mnemoniche, o da un uso di istruzioni che contrastano con le regole definite dall'assembler.
Supponiamo che il programma sorgente GIOCO.ASM si trovi nel drive B, che il Macro Assembler Microsoft sia nel drive A e che il drive B sia quello selezionato. Compare allora sullo schermo il prompt B>.
Per invocare l'assembler è sufficiente scrivere:

```
B>A:MASM
```

Tabella B.1 Opzioni del Macro Assembler Microsoft

Opzione	Descrizione
/A	Questa opzione (attiva per default) indica all'assembler di elencare i segmenti del codice sorgente in ordine alfabetico.
/B	Questa opzione permette di definire la dimensione del buffer del file, in kB. Il valore minimo (per default) è 32 kB, mentre il valore massimo è 63 kB (non 64 kB).
/C	Questa opzione informa l'assembler MASM di generare un file di cross-reference.
/D	Quando si verifica un errore di fase, è stata riscontrata una incompatibilità tra i valori generati dalla prima passata e quelli generati dalla seconda passata nel processo di traduzione. Per localizzare questa incongruenza, si può informare l'assembler di generare un listato durante la prima passata, utilizzando l'opzione /D. Questo listato

Tabella B.1 (continua)

Opzione	Descrizione
	addizionale può essere confrontato con quello che viene generato automaticamente dalla seconda passata.
/E	Questa opzione indica all'assembler di tradurre il codice sorgente per l'80287/80387 e di generare costanti in virgola mobile nella forma attesa dal coprocessore. Opera nello stesso modo dell'opzione /R.
/I	Questa opzione permette di definire i cammini di ricerca per i file inclusi. Utilizzando questa opzione per ogni cammino, è possibile definire fino a 10 cammini di ricerca.
/ML	Questa opzione informa l'assembler di considerare distinte variabili scritte in minuscolo da variabili con nome identico scritte in maiuscolo (ad esempio MEM_WORD viene considerata diversa da mem_word).
/MU	Questa opzione (attiva per default) informa l'assembler di convertire tutte le lettere minuscole in lettere maiuscole nei nomi Public o External.
/MX	Questa opzione informa l'assembler di distinguere tra lettere maiuscole e lettere minuscole nei nomi Public o External. Ad esempio, le variabili MEM_WORD e Mem_Word vengono tradotte nel codice oggetto in due forme distinte.
/N	Questa opzione informa l'assembler di non generare una tabella di simboli al termine del file. LST.
/P	Questa opzione (attiva solo sotto il controllo del microprocessore 80286/80386) informa l'assembler di controllare le eccezioni inaccettabili in modalità protetta 80286.
/R	Avviene la traduzione del codice sorgente in una forma compatibile con il coprocessore 80287/80387 per quanto riguarda i valori numerici prodotti. Il codice macchina generato con questa opzione non può essere eseguito correttamente su sistemi che non dispongono del coprocessore.
/S	Questo parametro disattiva l'opzione di default /A, per cui i segmenti vengono elencati nello stesso ordine con cui appaiono nel codice sorgente.
/T	Questa opzione disattiva la stampa del messaggio che informa l'utente della corretta terminazione dell'operazione di traduzione in codice macchina.
/V	Questa opzione (abbreviazione di Verbose) informa l'assembler di fornire informazioni aggiuntive durante l'operazione di traduzione relative al numero di linea e ai simboli.
/X	Questa opzione induce l'assembler a memorizzare nel file. LST una copia delle istruzioni che costituiscono il corpo di ogni direttiva IF valutata falsa.
/Z	Questa opzione è molto utile in quanto informa l'assembler MASM di visualizzare sullo schermo le linee di codice contenenti eventuali errori.

Sullo schermo vengono visualizzate le seguenti linee di testo:

```
Microsoft (R) MACRO Assembler   Version 4.00
Copyright (C) Microsoft Corp 1981, 1983, 1984, 1985. All rights
reserved.

Source filename [.ASM]
```

L'assembler richiede all'utente di specificare il nome del file sorgente che deve essere tradotto in codice macchina. Le parentesi quadre [ ] contengono l'estensione per default (.ASM) che viene assegnata al file sorgente dall'assembler. Poiché il codice sorgente GIOCO è stato salvato con l'estensione .ASM, scrivete da tastiera solo GIOCO e premete il tasto ENTER come risposta alla prima richiesta del calcolatore.
Si tenga presente che il Macro Assembler Microsoft permette di specificare le indicazioni del drive e del sottodirettorio (path name) se state utilizzando dischi fissi.
Dopo aver premuto il tasto ENTER, viene visualizzata una seconda richiesta:

```
Object filename [GIOCO.OBJ]:
```

L'assembler indica che, a meno di esplicita controindicazione, il nome del codice oggetto sarà GIOCO.OBJ. Premete il tasto ENTER se accettate questo identificatore per default e vedrete visualizzata sullo schermo una terza richiesta:

```
Source Listing [NUL.LST]:
```

Un file con l'estensione .LST (listato) costituisce il prodotto stampabile delle operazioni dell'assembler poiché contiene i numeri di linea, la traduzione equivalente in codice macchina di ogni istruzione e una tabella di simboli. L'assembler non genera questo file per default, come viene indicato dal mnemonico NUL.
Perché l'assembler produca il file .LST, dovete scrivere da tastiera GIOCO e premere il tasto ENTER. Non è necessario specificare da tastiera l'estensione .LST, in quanto questa viene assunta per default dall'assembler.
Dopo aver premuto il tasto ENTER, l'assembler visualizza sullo schermo la quarta e ultima richiesta:

```
Cross reference [NUL.CRF]:
```

L'assembler genera il file .CRF che verrà impiegato dall'utilità CREF.EXE per creare il listato finale di cross-reference (nel quale, accanto a ogni simbolo usato, compare l'elenco dei numeri di linea in cui tale simbolo è citato). Ancora una volta l'assembler non crea per default il file .CRF (mnemonico NUL), per cui, per ottenerlo, dovete specificare da tastiera il nome GIOCO

e premere il tasto ENTER. L'assembler allora crea il file GIOCO.CRF, che verrà utilizzato in seguito.
L'assembler ora esamina il file sorgente GIOCO.ASM e produce i file .OBJ, .LST e .CRF. Per assicurarsi che queste operazioni siano state eseguite correttamente, richiedete su video il direttorio del disco B:

```
GIOCO   ASM     758   12-27-85   6:47p
GIOCO   OBJ      57   12-29-85   2:55p
GIOCO   LST    1656   12-29-85   2:55p
GIOCO   CRF      69   12-29-85   2:55p
```

È noto che il file GIOCO.ASM contiene il codice ASCII del programma e che il file GIOCO.OBJ non può essere visualizzato, in quanto rappresenta la versione in codice macchina del programma. Anche il file GIOCO.CRF non può essere visualizzato, in quanto costituisce un passo intermedio nella generazione di una tabella di cross-reference. Il file che può invece essere visualizzato è GIOCO.LST.
Come abbiamo detto precedentemente, il file .LST contiene la versione tradotta del codice sorgente, l'indicazione dei numeri di linea, la rappresentazione equivalente in codice macchina delle istruzioni e una tabella di simboli. Questo file di solito viene consultato in fase di correzione e di analisi del programma e può essere stampato inviando da tastiera i comandi TYPE GIOCO.LST, CTRL-PRTSC e ENTER:

```
Microsoft (R) Macro Assembler  Version 4.00     12/29/85 02:55:37
                                                Page     1-1

                      ;per macchine 80286/80386
                      ;programma che campiona il joystick e restituisce i valori di
                      ;potenziometro nei registri indicati.


                      PAGE ,132                           ;dimensionamento pagina

      0000            CODICE     SEGMENT PARA 'CODICE'    ;definisce il segmento di codice
      0000            PROCEDURA  PROC    FAR              ;inizio della procedura
                                 ASSUME  CS:CODICE
      0000 1E                    PUSH    DS               ;salva DS sullo stack
      0001 2B C0                 SUB     AX,AX            ;azzera AX
      0003 50                    PUSH    AX               ;salva 0 sullo stack

                      ;campionamento potenziali del joystick. AX=A(X), CX=B(X), DX=B(Y)
                      ;dopo l'interruzione
      0004 B4 84                 MOV     AH,84H           ;campionamento leva di comando
      0006 BA 0001               MOV     DX,01H           ;campionamento potenziali
      0009 CD 15                 INT     15H

      000B CB                    RET                      ;il controllo ritorna al DOS
      000C            PROCEDURA  ENDP                     ;fine della procedura
      000C            CODICE     ENDS                     ;fine del segmento di codice

                                 END                      ;fine del programma
```

```
Microsoft (R) MACRO Assembler   Version 4.00        12/29/86 02:55:37
                                                    Symbols-1

Segment and Groups:

           Name                Size    Align    Combine    Class

CODICE.....................    000C    PARA     NONE       'CODICE'

Symbols:

           Name                Type    Value    Attr

PROCEDURA..................    F PROC  0000     CODICE     Length =000C

     25 Source  Lines
     25 Total   Lines
     24 Symbols

  24832 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors
```

OPZIONI

Il Macro Assembler Microsoft offre molte opzioni all'utente per tradurre un programma. Le opzioni vengono specificate dopo il comando e iniziano con una barra (/). Sono elencate in ordine alfabetico in Tabella B.1 (per maggiori dettagli, consultate il *Microsoft Macro Assembler Reference Manual*).

COMANDI IN FORMA ABBREVIATA

È possibile invocare l'assembler con comandi in forma abbreviata, che risultano particolarmente utili ai programmatori esperti.
Come abbiamo detto al Capitolo 2, quando viene scritto da tastiera:

```
B>A:MASM GIOCO
```

l'assembler immediatamente genera il file .OBJ, senza richiedere all'utente di specificare il nome che egli intende assegnare al file di codice oggetto. Inoltre, con il precedente comando, l'assembler non genera i file .LST e .CRF.

I seguenti comandi in forma abbreviata evitano al programmatore di dover rispondere alle richieste dell'assembler, in quanto vengono assegnati ai file .LST e .CRF gli identificatori per default.

```
B>A:MASM GIOCO,,,
```

Ogni virgola (,) disattiva una richiesta dell'assembler e attiva automaticamente per l'identificatore di file la codifica per default. Il comando NUL svolge la stessa funzione della virgola.
Le opzioni vengono specificate nel modo seguente:

```
B>A:MASM GIOCO/N
```

Questo comando informa l'assembler di non generare una tabella di simboli al termine del file .LST.

B.4 Cross-reference mediante CREF.EXE

Le informazioni di cross-reference sono molto utili in fase di correzione dei programmi, in quanto forniscono al programmatore un elenco in ordine alfabetico di dati, variabili, etichette, costanti e altri simboli che sono stati referenziati nel codice sorgente codice. Il listato include il numero di linea in corrispondenza del quale il simbolo è definito e l'indicazione di tutte le altre linee in cui il simbolo stesso è stato utilizzato.
Per generare un listato di cross-reference si deve selezionare la quarta opzione dell'assembler, che permette la generazione del file intermedio *nomefile*.CRF. Questo file viene utilizzato dal programma di utilità CREF.EXE per creare l'effettivo file di cross-reference.
Per generare il listato di cross-reference, scrivete semplicemente da tastiera:

```
B>CREF
```

e premete il tasto ENTER. Sullo schermo appare allora:

```
Microsoft (R) Cross-Reference Utility   Version 4.00
     Copyright (C) Microsoft Corp 1981, 1983, 1984, 1985. All right
     reserved

     Cross-reference [.CREF]
```

L'utilità CREF.EXE richiede il nome del file *nomefile*.CRF. Si risponda scrivendo da tastiera GIOCO e premendo il tasto ENTER. Si ricordi che non è indispensabile aggiungere l'estensione di file .CRF, poiché questa viene assegnata per default.
Dopo aver eseguito l'operazione, viene visualizzata sullo schermo la seguente richiesta:

```
List filename [GIOCO.REF]:
```

Il nome del file per default, che viene assegnato da CREF.EXE, è GIOCO.REF. Questo nome può essere modificato dall'utente, specificando – se necessa-

rio – un differente indicatore di drive. Se, invece, si accetta questo identificatore per default, è sufficiente premere il tasto ENTER.

CHIAMATA DI CREF.EXE IN FORMA ABBREVIATA

Come per il macro assembler, esiste un comando in forma ridotta per invocare l'utilità di cross-reference. Scrivendo da tastiera:

```
B>CREF GIOCO
```

si impone al programma CREF.EXE di cercare automaticamente il file GIOCO.CRF e poi di generare un file .REF con lo stesso nome. Nel nostro esempio, quindi, questo file si chiama GIOCO.REF.
Se questa operazione ha luogo senza errori, sullo schermo viene visualizzata l'indicazione del drive selezionato (prompt). Per assicurarsi che il file sia stato effettivamente creato, è sufficiente richiedere il direttorio del dischetto B: nel nostro caso, deve comparire il file GIOCO.REF. Se scrivete da tastiera TYPE GIOCO.REF, viene visualizzato sullo schermo il listato di cross-reference:

```
B>TYPE GIOCO.REF


Microsoft Cross-reference   Version 4.00            Mon Dec 02
02:35:00  1985


Symbol Cross Reference                          (# is definition)
Cref-1

CODICE . . . . .                 8

CODICE . . . . . . .             8#    10    23
PROCEDURA. . . . . . .           9#    22

3 Symbols

63048 Bytes Free
```

Da questo listato è possibile determinare il numero di segmenti che sono stati definiti nel programma. Nel nostro caso, esiste un solo segmento: CODICE. Si possono anche notare le dichiarazioni delle entità simboliche CODICE e PROCEDURA. Il carattere #, che segue un numero di linea, indica che il relativo simbolo (ad esempio CODICE o PROCEDURA) è stato definito su quella linea. I numeri di linea seguenti specificano invece le linee in cui il simbolo è stato referenziato.
Anche in questo breve esempio, è possibile utilizzare il listato di cross-reference per assicurarsi che ogni sezione di codice sia stata conclusa con

la direttiva END, come nel caso di CODICE (linea 23) e PROCEDURA (linea 22). Aumentando la dimensione del programma scritto in linguaggio assemblatore, cresce l'importanza delle informazioni di cross-reference, per garantire una rapida correzione del codice sorgente.
Si può ottenere una copia di questo file semplicemente accendendo la stampante e inviando da tastiera il comando TYPE seguito dal nome del file, come viene qui di seguito indicato:

```
B>TYPE GIOCO.REF
```

Si prema CTRL-PRTSC e poi il tasto ENTER. Premete nuovamente CTRL-PRTSC per terminare la stampa.
Sia che abbiate generato il listato di cross-reference, sia che non lo abbiate generato, il passo successivo consiste nel risolvere i riferimenti esterni presenti nel file .OBJ. Questa operazione viene eseguita invocando il programma di utilità LINK.EXE che viene fornito insieme all'assembler.

B.5 Collegamento: LINK.EXE

Il linker (LINK.EXE) esamina il file .OBJ e genera un programma rilocabile. In questo modo, il sistema operativo viene abilitato a caricare in una qualunque area di memoria disponibile la versione eseguibile del programma scritto in linguaggio assemblatore. L'alternativa alla coesistenza di più programmi in memoria è di definire per il programma indirizzi fisici prestabiliti. Supponiamo che il programma inizi dalla locazione di memoria 0100H e che il risultato delle elaborazioni sia stato memorizzato nella locazione 4F3AH. Se un altro programma referenzia la stessa locazione, si crea un problema di condivisione di memoria. Un programma rilocabile, invece, può essere allocato in memoria in corrispondenza di locazioni libere poiché tutti gli indirizzi si modificano automaticamente in fase di esecuzione, grazie alle modifiche apportate dal linker al codice oggetto.
Il linker esegue altre operazioni estremamente utili, come ad esempio quella di collegare file di tipo .OBJ, che sono stati compilati separatamente. (È possibile così sezionare in moduli più piccoli un programma in linguaggio assemblatore di grandi dimensioni). Il linker può anche risolvere i riferimenti tra il programma principale e le routine chiamate appartenenti a librerie esterne.
Per invocare il linker, inviate da tastiera il comando LINK. Saranno allora visualizzate le seguenti linee di testo:

```
Microsoft (R) 8086 Object Linker  Version 3.05
Copyright (C) Microsoft Corp 1983, 1984, 1985. All rights
reserved.

Object Modules [.OBJ]:
```

Il linker richiede all'utente il nome del file .OBJ da collegare.
Nel nostro esempio, si scriva da tastiera GIOCO. Ancora una volta, non è necessario specificare l'estensione di file .OBJ, in quanto il linker la assume per default. Viene allora visualizzata la seguente richiesta:

```
Run file [GIOCO.EXE]:
```

Per default, il linker fornisce il nome della versione eseguibile del programma. È possibile sostituire tale nome con un altro, senza dover specificare l'estensione .EXE.
Si prema il tasto ENTER se, invece, intendete accettare questo identificatore di file. La successiva richiesta visualizzata sullo schermo risulta:

```
List File [NUL.MAP]:
```

Il file .MAP contiene l'elenco dei segmenti che sono stati definiti nel codice sorgente e specifica i valori di offset presenti nel file eseguibile, per cui si rivela molto utile in fase di correzione del programma.
Il linker non genera questo file per default (come viene indicato dal mnemonico NUL contenuto tra le parentesi quadre) ma, per ottenerlo, dovete specificare il nome GIOCO e poi premere il tasto ENTER. Da ultimo, viene visualizzata la quarta richiesta:

```
Libraries [.LIB]:
```

Il linker chiede ora il nome delle librerie contenenti le routine che sono referenziate nel codice sorgente. Nel nostro esempio, non è stata utilizzata alcuna libreria definita dall'utente, per cui è sufficiente premere il tasto ENTER per andare avanti.
A questo punto appare il seguente messaggio:

```
Warning: No STACK segment
```

Questo messaggio indica che il codice sorgente del nostro programma non contiene la definizione di un segmento di stack. In questo caso, il messaggio può essere ignorato in riferimento al tipo di file che si intende creare. Il programma di esempio GIOCO.ASM è stato scritto per essere convertito in un file estremamente compatto chiamato file .COM.
Richiediamo ora il direttorio del disco B, per assicurarci che ogni passo del ciclo di sviluppo dell'applicazione sia stato compiuto correttamente:

```
GIOCO   ASM    758   12-27-85   6:47p
GIOCO   OBJ     57    1-02-86   6:36p
GIOCO   LST   1656    1-02-86   6:36p
GIOCO   CRF     69    1-02-86   6:36p
GIOCO   REF    268    1-02-86   6:36p
```

```
GIOCO    MAP      154      1-02-86    7:13p
GIOCO    EXE      524      1-02-86    7:13p
     7 File(s)    4159488 bytes free
```

A questo punto potete lanciare il file GIOCO.EXE che costituisce la versione rilocabile, eseguibile del programma GIOCO.
Prima di convertire questo file in un file .COM, discutiamo alcune opzioni del linker e indichiamo il significato di alcuni comandi espressi in forma abbreviata.

OPZIONI DEL LINKER E COMANDI IN FORMA ABBREVIATA

Le opzioni elencate in Tabella B.2 possono essere incluse nel comando di invocazione del linker, facendole precedere dal carattere barra (/).
Come per il macro assembler, esistono alcune forme abbreviate di comandi che possono essere utilizzate per invocare il linker e che vengono elencate in Tabella B.3.

CONFRONTO TRA FILE .EXE E FILE .COM

I vantaggi di un file .EXE sono due: innanzitutto, il file è rilocabile, per cui più di un programma può essere caricato in memoria sfruttando lo spazio disponibile; secondariamente, i file di tipo .EXE permettono di utilizzare fino a quattro segmenti: STACK, DATI, CODICE e EXTRA. In questo modo, è possibile raggiungere una buona modularità del codice e creare programmi di grandi dimensioni. Ogni segmento presente in un file di tipo .EXE può avere una dimensione massima di 64 kB. Lo svantaggio di un file .EXE è costituito dalla quantità di codice aggiunto dal linker per rendere possibile la rilocazione. Nel nostro esempio, la dimensione del file contenente il programma GIOCO.EXE è 524 byte.
Al contrario, un file .COM può contenere un solo segmento di 64 kB, in cui risiedono le informazioni relative allo STACK, ai DATI e al CODICE. Quando la dimensione di un programma in linguaggio assemblatore non è elevata, 64 kB risultano più che sufficienti per includere tutti i dati, le istruzioni e le operazioni di gestione dello stack. Diversamente dal file .EXE, il file .COM non è rilocabile e deve iniziare all'indirizzo 0100H. Il vantaggio di un file .COM rispetto ad un file .EXE è che il file .COM riduce drasticamente la dimensione del file eseguibile. Per creare un file .COM, viene fornita dal DOS l'utilità EXE2BIN.EXE.

Tabella B.2 Opzioni del Linker

Opzione	Descrizione
/HELP	Questa opzione (abbreviazione /HE) informa il linker di visualizzare sullo schermo un elenco delle opzioni disponibili.
/DS ALLOCATION	Questa opzione (abbreviazione /DS) informa il linker di caricare tutti i dati definiti nell'area dati in corrispondenza dell'estremità superiore di quell'area. Per default, il dato viene caricato in corrispondenza dell'estremità inferiore dell'area, iniziando dall'offset 0.
/EXEPACK	Questa opzione (abbreviazione /E) indica al linker LINK di rimuovere sequenze di byte che si ripetono, per minimizzare la dimensione della tabella di rilocazione. Comunque, questa opzione può in realtà incrementare la dimensione del file e viene utilizzata al meglio solo su codice sorgente contenente molte sequenze di caratteri ripetute.
/HIGH	Questa opzione (abbreviazione /H) informa il linker di caricare il file eseguibile nella zona di memoria superiore, senza interferire con il codice COMMAND.COM.
/LINENUMBERS	/L o /LINE informa il linker di includere nel file. LST gli indirizzi e i numeri di linea di ognuna delle istruzioni sorgenti dei moduli di ingresso.
/MAP	L'opzione /M indica al linker LINK di generare un elenco di tutti i simboli Public presenti nel codice sorgente.
/NOIGNORECASE	Questa opzione (abbreviazione /NOI) fa sì che il linker distingua tra carattere minuscolo e carattere maiuscolo. Questa opzione può essere utilizzata insieme all'opzione /ML o /MX affinché l'assembler distingua le variabili Public scritte in caratteri minuscoli da quelle identiche scritte in caratteri maiuscoli.
/NODEFAULT LIBRARY SEARCH	Questa opzione (abbreviazione/NOD) informa il linker di non richiedere all'utente di specificare le librerie.
/NOGROUP ASSOCIATION	Questa opzione (abbreviazione /NOG) informa il linker di correggere gli indirizzi esterni di tipo long, anche se il simbolo è stato definito in un segmento interno ad un'area definita.
/PAUSE	/P informa il linker di generare su video un messaggio che richieda l'inserimento nel drive di un dischetto per contenere il file eseguibile.
/STACK:*size*	/S seguita da un valore decimale compreso tra 0 e 65 536 permette al programmatore di specificare la dimensione del segmento stack. I valori compresi tra 0 e 512 assegnano una quantità minima di stack di 512 byte.
/CPARMAXALLOC	Questa opzione (abbreviazione /C) definisce il massimo numero di paragrafi a 16 byte che occorrono ad un programma quando viene caricato in memoria.

Tabella B.3 Comandi in forma abbreviata per il linker

Comando	Descrizione
LINK *nomefile;*	Questo comando in forma abbreviata invoca il linker, che utilizza il file *nomefile*.OBJ in ingresso e automaticamente identifica il file di uscita con *nomefile*.EXE.
LINK *nomefile,,;*	Questo comando in forma abbreviata indica al linker di cercare il file *nomefile*.OBJ in ingresso e di generare i file di uscita *nomefile*.EXE e *nomefile*.MAP.

Nota: Ogni virgola (,) disattiva una richiesta del linker e attiva il nome per default del file corrispondente a quella particolare opzione. Il nome NUL può essere sostituito alla virgola per disattivare una particolare opzione.

B.6 Creazione dei file .COM

Il programma di utilità EXE2BIN.EXE genera la versione .COM da un file .EXE. Per invocare questa utilità, scrivete da tastiera:

```
B>A:EXE2BIN GIOCO GIOCO.COM
```

e poi premete il tasto ENTER. Supponiamo che il programma EXE2BIN.EXE si trovi sul dischetto inserito nel drive A, per cui basta scrivere A:EXE2BIN per referenziare il programma. Il file di tipo .EXE da convertire è GIOCO (si noti che non è indispensabile specificare l'estensione .EXE), mentre il nome del file di tipo .COM da creare è GIOCO.COM. Si ricordi, in questo caso, di specificare l'estensione di file, perché altrimenti il programma EXE2BIN adotta l'estensione per default .BIN, che non rappresenta un'estensione corretta per un file eseguibile (cambiatela eventualmente con il comando Rename).
Confrontiamo ora la dimensione in byte dei file GIOCO.EXE e GIOCO.COM:

```
GIOCO   EXE    524   1-02-86   7:45p
GIOCO   COM     12   1-02-86   7:45p
```

Si noti l'enorme riduzione di codice ottenibile con la versione di tipo .COM.

C

Turbo Editasm

Questa appendice vi permetterà di acquisire immediatamente le necessarie conoscenze per utilizzare il Turbo Editasm. In particolare, vengono discussi i passi che occorrono per creare un programma in linguaggio assemblatore e per tradurre – tramite il linker – il codice sorgente in un file eseguibile di tipo .COM oppure di tipo .EXE. La Figura C.1 mostra le due diverse modalità di sviluppo di un programma in linguaggio assemblatore.

C.1 Informazioni di carattere generale

Il Turbo Editasm possiede un editor di video. Diversamente dai Macro Assembler IBM e Microsoft, che richiedono un editor di testi ASCII separato, il Turbo Editasm incorpora l'editor nell'assembler. In questo modo, l'utente può richiamare istantaneamente – durante la traduzione – la linea di codice in cui è stato scoperto un errore sintattico. Il Turbo Editasm incorpora anche molte altre funzioni (come ad esempio un programma di utilità di cross-reference), esegue una traduzione linea per linea in codice macchina e genera una tabella di simboli.
Inoltre, il Turbo Editasm supporta il set di istruzioni, in modalità di indirizzamento reale e protetta, del microprocessore 80286, supporta il set di istruzioni dei coprocessori matematici 8087 e 80287, oltre a cinque tipi di dati in virgola mobile, interi BCD ed estesi, e garantisce la possibilità di coesistenza nello stesso file sorgente di tipi di dati Microsoft e 8087.
Questa appendice prende in considerazione il Turbo Editasm TASMB, che

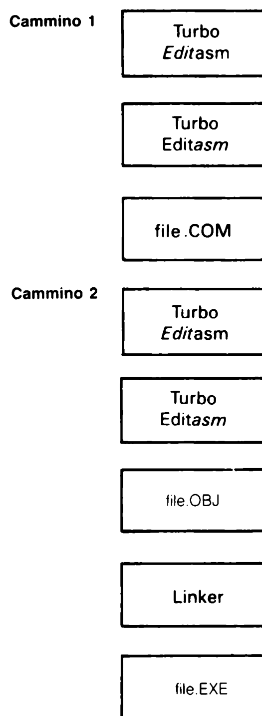


Figura C.1 Due modalità di sviluppo di un programma in linguaggio assemblatore

permette di generare sia file .COM che file .EXE e che utilizza l'editor di video residente.
Se durante la traduzione del programma, la posizione del segmento dati (precedente al segmento di codice) crea qualche problema, trasferite il segmento dati dopo il segmento di codice: Turbo Editasm fa un controllo dei tipi di dati più severo del Macro Assembler IBM.

C.2 Creazione di codice sorgente nel linguaggio assemblatore

Il ciclo di sviluppo di una applicazione in linguaggio assemblatore ha inizio con la definizione di un problema da risolvere a livello macchina. Ad esempio, può essere necessario visualizzare sullo schermo un messaggio.

Prima di pensare alla soluzione programmativa, è indispensabile conoscere le modalità con cui il calcolatore manipola le stringhe di caratteri. In questo caso, il programmatore può invocare una interruzione DOS, come abbiamo già sottolineato in uno dei capitoli precedenti.
La conoscenza di questa informazione garantisce la scrittura corretta di codice sorgente. Come abbiamo discusso nel Capitolo 2, il programma sorgente utilizza la codifica mnemonica per rappresentare simbolicamente le istruzioni in codice macchina e deve soddisfare le regole grammaticali (sintassi) dell'assembler, perché la traduzione proceda correttamente.
Per scrivere il codice sorgente utilizzando il Turbo Editasm, inserite il dischetto contenente il Turbo Editasm – nel nostro caso, si tratta del TASMB – nel drive A (drive selezionato), il dischetto di lavoro nel drive B e scrivete da tastiera:

TASMB

A questo punto appaiono sullo schermo le seguenti informazioni:

```
-------------------------------------
TURBO EDITASM              Ver 1.03B
                                PC-DOS

    Copyright (C) 1984, 1985 by SPEEDWARE
    -------------------------------------

    Assem Source  Edit Source  Get Source  Write Source

    Run Codefile  Hexdump File Kill File   List File

    Symbol List   Xrefer List  Directory   New Drive - Directory

    Asm Options   Value        Quit

    65278 Byte(s) Available
        0 Byte(s) Used

    (A)
```

Le lettere in neretto rappresentano i comandi in forma abbreviata che selezionano le opzioni del Turbo Editasm TASMB. In fondo allo schermo, è visualizzato il prompt di TASMB – cioè (**A**) – ad indicare che il sistema è in attesa di un comando.
Premete la lettera E (**E**dit Source): lo schermo viene allora cancellato e appare la seguente informazione in alto a destra, ad indicare che siete in modalità Edit.

Line 1   Col 1   Insert

Scrivere da tastiera il seguente codice:

```
PAGE ,132                        ;dimensionamento pagina
         SEGMENT  PARA 'DATI'
```

```
DATI      DB      'The 80286/80386 are powerful microprocessors'
MESS      DB      '$'
                  ENDS
DATI
                  SEGMENT  PARA 'CODICE'  ;definisce il segmento di codice
CODICE            PROC     FAR            ;inizio della procedura
PROCEDURA         ASSUME   CS:CODICE,DS:DATI
                  PUSH     DS             ;salva DS sullo stack
                  SUB      AX,AX          ;azzera AX
                  PUSH     AX             ;salva 0 sullo stack
                  MOV      AX,DATI        ;indirizzo di DATI in AX
                  MOV      DS,AX          ;indirizzo di DATI in DS
                  LEA      DX,MESS        ;indirizzo di MESS in DX
                  MOV      AH,9H          ;parametro DOS
                  INT      21H            ;chiamata di interruzione

                  RET                     ;il controllo ritorna al DOS
PROCEDURA         ENDP                    ;fine della procedura
CODICE            ENDS                    ;fine del segmento di codice

                  END                     ;fine del programma
```

I comandi di editor di TASMB sono simili a quelli del word processor WordStar. È possibile utilizzare i tasti di movimento cursore e i tasti INS e DEL per scrivere il testo.
Salvate su dischetto quanto avete scritto su video. Premendo il tasto funzione F2 appare in alto a sinistra sullo schermo la seguente informazione:

Write Source File:

Scrivete quindi da tastiera:

B:PROGEXE

e premete il tasto ENTER. TASMB allora associa automaticamente al file l'estensione .ASM.
Per tradurre il programma in codice macchina, uscite dall'editor premendo CTRL-K-D. TASMB vi risponderà con il prompt (**A**). Premete il tasto A (**Assem**ble), per iniziare la traduzione, e apparirà la seguente richiesta:

Use File: B:PROGEXE.OBJ (Y/N)?

Premete il tasto Y per creare il file .OBJ con lo stesso nome del file sorgente. TASMB è un assembler a due passate e l'indicazione di quale passata sia in esecuzione viene visualizzata con il seguente messaggio:

Assembling
Pass One

Pass Two

Supponendo che non ci siano errori in fase di traduzione, vengono visualiz-

zate sullo schermo le seguenti informazioni:

```
25 Source Line(s), No Assembly Error(s).
12 Object Byte(s),54716 Byte(s) Free.
Assembly Time: 1 second(s)

(A)
```

C.3 Opzioni di traduzione

TASMB fornisce un certo numero di opzioni che risultano estremamente facili da utilizzare. Selezionando il tasto o (Asm **O**ptions) dal menu principale, vengono visualizzate le opzioni disponibili, come ad esempio la stampa del codice tradotto, la generazione di una tabella di cross-reference e di simboli, la visualizzazione del codice sullo schermo e la memorizzazione su disco. Alcuni dei valori ON/OFF di default relativi alle opzioni possono comparire differenti da quelli indicati):

```
* Editasm Options *

  F1 - Screen    (ON)
  F2 - Printer   (OFF)
  F3 - Symbols   (OFF)
  F4 - Xrefer.   (OFF)
  F5 - Memory    (OFF)
  F6 - ErrWait   (OFF)
  F7 - OBJ File  (OFF)
  F8 - COM File  (OFF)
  F9 - LST File  (OFF)
  F10- Undefd.   (OFF)

Select option or <CR> to Exit.
```

C.4 Creazione di file .OBJ

La creazione del file .OBJ costituisce un passo intermedio necessario per preparare il codice sorgente all'esame del linker, che genera un file eseguibile .EXE. Quest'ultimo file contiene codice macchina aggiuntivo che lo rende rilocabile, cioè tale da essere allocato in una qualunque zona di memoria disponibile.

Per creare il file .OBJ, selezionate l'opzione Asm Options dal menu principale di TASMB e poi attivate l'opzione .OBJ, premendo il tasto funzione F7. Premendo il tasto ENTER, si ritorna al menu principale e qui potete selezionare l'opzione Assem per tradurre il codice sorgente in codice oggetto.

TASMB vi chiede a questo punto di specificare il nome del file da utilizzare:

B:\PROGEXE.OBJ (Y/N)?

Premete il tasto Y se volete accettare questo nome di default.
Per verificare che il file .OBJ è stato generato correttamente, selezionate l'opzione Directory del menu principale per richiedere la visualizzazione del direttorio del disco. Viene allora visualizzato sullo schermo:

PROGEXE.ASM
PROGEXE.OBJ

Il prossimo passo da compiere è quello di risolvere i riferimenti esterni presenti nel file .OBJ utilizzando il Linker che è memorizzato nel disco di sistema (vedere a questo proposito il Paragrafo C.7).

C.5 Creazione di un file .LST

Per facilitare il processo di correzione del codice, è utile spesso disporre di un listato del codice tradotto. Per creare questo listato, basta selezionare l'opzione Asm Options del menu principale e successivamente premere il tasto funzione F9 (vedere la Tabella C.1 per maggiori dettagli). A questo punto si abbandoni l'opzione Asm Options premendo il tasto ENTER. Si selezioni l'opzione Assem per eseguire la traduzione del codice e TASMB visualizzerà sullo schermo una doppia richiesta: una relativa al nome del file tradotto

Use Object File: B:\PROGEXE.OBJ (Y/N)?

e una relativa al nome da utilizzare per il file .LST

Use File: B\PROGEXE.LST (Y/N)?

Per analizzare il contenuto del file .LST è possibile utilizzare l'editor TASMB o visualizzare il file tramite il comando DOS TYPE. Volendo una copia su stampante, basta premere il tasto funzione F2 prima di generare il listato. Per visualizzare il file tradotto su schermo, premete invece il tasto funzione F1.
Nel caso del programma PROGEXE.ASM, il listato tradotto assume il seguente formato:

```
TURBO ASSEMBLER      March 2,1986    02:41:57 p.m.        Page 1

   1                    PAGE ,132                       ;dimensionamento pagina
```

Tabella C.1 Opzioni dell'assembler

Tasto	Descrizione opzione
F1	SCREEN – Attivando l'opzione F1 viene visualizzato sullo schermo il codice sorgente mentre viene tradotto. Se F1 è OFF, vengono visualizzati sullo schermo solo gli errori trovati in fase di traduzione. L'assembler, in questo caso, opera più velocemente, in quanto non deve fermarsi per visualizzare ogni linea tradotta sullo schermo.
F2	PRINTER – Quando questa opzione è attiva, viene inviata su stampante una copia del codice tradotto. TASMB non può inviare simultaneamente il codice tradotto sullo schermo e sulla stampante.
F3	SYMBOLS – Questa opzione informa l'assembler di includere una tabella di simboli alla fine del listato del codice corgente tradotto. La tabella viene inviata all'unità periferica selezionata come definito in F1 e F2.
F4	XREF – Attivando l'opzione F4, l'assembler genera le informazioni di cross-reference, che consistono nell'elencare ogni simbolo utilizzato nel codice sorgente seguito dalla linea o dalle linee di codice sorgente in cui il simbolo stesso è stato referenziato. Queste informazioni vengono inviate all'unità periferica selezionata come definito in F1 e F2.
F5	MEMORY – Con F5 nella posizione ON, TASMB alloca in memoria direttamente la versione in codice macchina del programma, che può essere eseguita selezionando dal menu principale di TASMB il comando Run (*Nota*: con questa opzione attiva, l'istruzione INT 20h di ritorno al TASMB può essere inserita in un qualunque punto del programma sorgente).
F6	ERRWAIT – È una opzione molto utile per listati lunghi, in quanto informa l'assembler di interrompere l'esecuzione ad ogni errore che trova nel listato sorgente. Durante questa pausa, il programmatore può referenziare la linea di codice e correggere l'errore (premendo il tasto ESC) oppure continuare la traduzione premendo il tasto ENTER.
F7	OBJ FILE – Questa opzione permette la creazione di un file .OBJ (come abbiamo descritto nel paragrafo sulla creazione di file .OBJ).
F8	COM FILE – Questa opzione permette la creazione del file .COM (come abbiamo descritto nel paragrafo sulla creazione di file .COM).
F9	LST FILE – Questa opzione permette di stabilire se viene creato un file .LST e se, oppure no, il file include una tabella dei simboli o le informazioni di cross-reference. Quando F9 è in posizione attiva (ON), TASMB richiede il nome da assegnare al file .LST. Ad esempio, per il programma PROGEXE.ASM, TASMB visualizza: 'Use File: PROGEXE. LST (Y/N)?'. Se l'opzione F3 (tabella di simboli) è attiva, il file .LST contiene in fondo la tabella di simboli, mentre se è attiva l'opzione F4, contiene le informazioni di cross-reference.
F10	Non utilizzata correntemente

```
  0000                    DATI      SEGMENT  PARA 'DATI'
  0000 54 68 65 20 38 30  MESS      DB       'The 80286/80386 are powerful microprocessors'
       32 38 36 2F 38 30
       33 38 36 20 61 72
       65 20 70 3F 77 65
       72 66 75 6C 20 6D
       69 63 72 6F 70 72
       6F 63 65 73 73 6F
       72 73
  002C 24                           DB       '$'
  002D                    DATI      ENDS

  0000                    CODICE    SEGMENT  PARA 'CODICE'  ;definisce il segmento di codice
  0000                    PROCEDURA PROC     FAR            ;inizio della procedura
                                    ASSUME   CS:CODICE,DS:DATI
  0000 1E                           PUSH     DS             ;salva DS sullo stack
  0001 29 C0                        SUB      AX,AX          ;azzera AX
  0003 50                           PUSH     AX             ;salva 0 sullo stack
  0004 B8 ----                      MOV      AX,DATI        ;indirizzo di DATI in AX
  0007 8E D8                        MOV      DS,AX          ;indirizzo di DATI in DS

  0009 8D 16 0000                   LEA      DX,MESS        ;indirizzo di MESS in DX
  000D B4 09                        MOV      AH,9H          ;parametro DOS
  000F CD 21                        INT      21H            ;chiamata di interruzione

  0011 CB                           RET                     ;il controllo ritorna al DOS
  0012                    PROCEDURA ENDP                    ;fine della procedura
  0012                    CODICE    ENDS                    ;fine del segmento di codice

                                    END                     ;fine del programma

25 Source Line(s), No Assembly Error(s).
12 Object Byte(s),54716 Byte(s)  Free.
Assembly Time: 1 second(s)
```

C.6 Creazione di tabelle di simboli e informazioni di cross-reference

Come abbiamo già sottolineato, l'opzione Asm Options del TASMB permette di elencare le opzioni che sono disponibili. Tra queste ci sono le opzioni per la creazione di una tabella di simboli e delle informazioni di cross-reference che risultano particolarmente utili in fase di correzione di un programma. In questo modo, viene elencato ogni simbolo presente nel codice sorgente e le linee in corrispondenza delle quali viene utilizzato. Dopo aver attivato l'opzione Asm Options, è sufficiente premere il tasto funzione F3 (tabella di simboli) oppure il tasto F4 (informazioni di cross-reference). La tabella di simboli per il programma PROGEXE.ASM risulta la seguente:

```
Assembling
Pass One                                        02:51:39
Pass Two
```

```
p.m

Segs & Groups:

        N a m e       Size    Align   Line   Combine    Class

CODICE. . . . . . *   0012    PARA    8      NONE       'CODICE'
DATI. . . . . . .     002D    PARA    3      NONE       'DATI'

Symbols:

        N a m e       Type    Value   Line   Attr

MESS. . . . . . .     Byte    0000    4      DATI
PROCEDURA . . . . *   F Proc  0000    9      CODICE     Length = 0012

25 Source Line(s), No Assembly Error(s).
12 Object Byte(s),54716 Byte(s)  Free.
Assembly Time: 1 second(s)

(A)
```

Per ottenere le informzioni di cross-reference, selezionate dal menu delle opzioni di TASMB l'opzione **X**refer List, premendo il tasto x:

```
        N a m e          Cross Reference(s):

MESS . . . . . . . .  4,17
CODICE . . . . . . .  8,23
DATI . . . . . . . .  3,6,14
PROCEDURA. . . . . .  9,22


    6 Symbol(s) Used.
28672 Byte(s)  for  Symbols.
28671 Byte(s)  for  Workarea.
```

C.7 Creazione di file .EXE

Ogni Linker DOS compatibile può essere utilizzato per convertire un file .OBJ nella versione eseguibile. Il Linker aggiunge il codice necessario perché il sistema possa allocare il programma in una qualunque zona di memoria disponibile.

Per utilizzare il Linker, è indispensabile uscire dal TASMB premendo il tasto Q – con il menu principale attivo – e inserire nel drive A il dischetto contenente il Linker. Supponendo che il file di codice oggetto .OBJ si trovi nel drive B e che il drive A sia quello selezionato, scrivete:

```
A>LINK
```

Il Linker risponde nel modo seguente:

```
IBM Personal Computer Linker
Version 2.30 (C) Copyright IBM Corp. 1981, 1985

Object Modules [.OBJ]:
```

Inserite il nome del file .OBJ che intende sottoporre al Linker. Nel nostro esempio, è possibile scrivere B:PROGEXE, senza dover specificare anche l'estensione .OBJ, in quanto questa viene assunta per default. Il Linker risponde con tre richieste:

```
Run File [B:PROGEXE.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
```

Premete semplicemente il tasto ENTER, se intendete accettare i parametri di default (vedere l'Appendice A e l'Appendice B per maggiori dettagli su come operano i Linker IBM e Microsoft).
Viene ora visualizzato il seguente messaggio di errore:

```
Warning: no stack segment
```

Questo messaggio avverte che nel programma non è stato definito il segmento di stack, ma per la nostra applicazione ciò non costituisce un errore.
A questo punto, è stato creato un file eseguibile .EXE. Richiedendo il direttorio del disco B, appare su video:

```
PROGEXE.ASM
PROGEXE.OBJ
PROGEXE.EXE
```

Per eseguire il file .EXE presente nel drive B, è sufficiente scrivere da tastiera il nome del programma.
Ad esempio, nel nostro caso:

```
A>B:PROGEXE
```

Compare sullo schermo allora il messaggio 'The 80286/80386 are powerful microprocessors'.

C.7 Creazione di file .COM

Prima di generare un file di tipo .COM, il codice sorgente deve avere un formato opportuno. Si consideri ancora il file PROGEXE.ASM, utilizzato per creare il file .EXE, che contiene due segmenti distinti, uno per i dati e uno per il codice. Questa segmentazione del programma potrebbe eludere la restrizione sulla dimensione massima di 64 Kbyte definita per i file .COM su macchine 80286.
Il codice sorgente del prossimo programma può essere tradotto in un file .COM per i microprocessori 80286 e 80386, in quanto contiene un solo segmento. Un file .COM viene eseguito ad una velocità maggiore della versione equivalente .EXE, per due ragioni: è presente un solo segmento, per cui non devono essere eseguiti calcoli di indirizzi tra segmenti distinti, che aumentano il tempo di esecuzione, e la versione .COM non è rilocabile, per cui non contiene informazioni di rilocazione aggiuntive (questo diminuisce la dimensione di codice e contribuisce ad aumentare la velocità di esecuzione).
Il seguente codice sorgente, PROCCOM.ASM, soddisfa le restrizioni che occorrono per generare un file .COM. Utilizzando l'editor di video di TASMB come precedentemente descritto, inserite da tastiera il codice PROGCOM.ASM esattamente come qui di seguito riportato:

```
        LEA     DX,MESS         ;indirizzo di MESS in DX
        MOV     AH,9H           ;parametro DOS
        INT     21H             ;chiamata di interruzione

        INT     20H             ;il controllo ritorna al DOS

MESS    DB      'The 80286/80386 are powerful microprocessors'
        DB      '$'

        END                     ;fine del programma
```

Ci sono meno linee di codice in PROGCOM.ASM rispetto alla versione PROGEXE.ASM in quanto per un file .COM non sono necessarie le dichiarazioni di segmento.
Per creare un file eseguibile .COM da un codice sorgente correttamente strutturato, selezionate l'opzione Asm Options dal menu principale delle opzioni di TASMB:

```
* Editasm Options *

  F1 - Screen     (ON)
  F2 - Printer    (OFF)
  F3 - Symbols    (OFF)
  F4 - Xrefer.    (OFF)
  F5 - Memory     (OFF)
  F6 - ErrWait    (OFF)
  F7 - OBJ File   (OFF)
```

```
    F8 - COM File  (OFF)
    F9 - LST File  (OFF)
    F10- Undefd.   (OFF)

Select option or <CR> to Exit.
```

Attivate l'opzione .COM File premendo il tasto di funzione F8 e poi il tasto ENTER. Assicuratevi di non premere il tasto di funzione F7, in quanto PROGCOM.ASM non presenta un formato corretto per generare un file .EXE. Ora, premete l'opzione Assem per attivare l'assembler e apparirà sullo schermo il seguente messaggio:

```
Use File: B:PROGCOM.COM (Y/N)? Y
```

Inserite da tastiera la lettera Y, se intendete accettare il nome di default per il file in uscita, e apparirà sullo schermo la seguente informazione:

```
Assembling
Pass One

Pass Two


11 Source Line(s), No Assembly Error(s).
55 Object Byte(s),54783 Byte(s) Free.
Assembly Time: 1 second(s)
```

Richiedete il direttorio del disco B:

```
PROGCOM.ASM
PROGCOM.COM
```

Per eseguire il file .COM, è sufficiente scrivere da tastiera il nome del programma, dopo essere usciti da TASMB. Nel nostro esempio, per eseguire PROGCOM.COM, scrivete PROGCOM e verrà visualizzato sullo schermo il seguente messaggio, in corrispondenza della posizione corrente del cursore: 'The 80286/80386 are powerful microprocessors'.

C.8 Altre opzioni dell'assembler TASMB

Dai precedenti esempi, si è potuto constatare che l'elenco di opzioni Asm Options di TASMB è molto ricco. Per visualizzare questo elenco (alcune dei valori di inizializzazione ON/OFF possono essere differenti), è sufficiente che sia attivo il menu principale e che inseriate da tastiera la lettera O:

```
* Editasm Options *

  F1 - Screen    (ON)
```

```
F2 - Printer   (OFF)
F3 - Symbols   (OFF)
F4 - Xrefer.   (OFF)
F5 - Memory    (OFF)
F6 - ErrWait   (OFF)
F7 - OBJ File  (OFF)
F8 - COM File  (OFF)
F9 - LST File  (OFF)
F10- Undefd.   (OFF)

Select option or <CR> to Exit.
```

La Tabella C.1 elenca ogni opzione e spiega il significato e le modalità di utilizzo.

D

Library Manager e librerie

Il Macro Assembler IBM (a partire dalla versione 2.0) e il Macro Assembler Microsoft (a partire dalla versione 3.0) contengono un gestore di libreria (Library Manager) che permette di definire e manipolare librerie create dall'utente o librerie di sistema. Questa appendice presenta un esempio che illustra come aggiungere una procedura ad una libreria preesistente. Nel Capitolo 7, sono state presentate alcune semplici modalità di gestione di una libreria, per cui è bene rileggere quel capitolo prima di affrontare la lettura di questa appendice.

D.1 Formattazione di codice

Nel Capitolo 9, la libreria di utilità IBM IBMUTIL.LIB ha permesso di formattare i risultati prodotti dal coprocessore 80287. Questa libreria contiene molte routine di conversione, una delle quali ($I8_OUTPUT) è stata utilizzata negli esempi presentati in quel capitolo. Poiché $I8_OUTPUT non restituisce i risultati nel formato richiesto, nel programma di pagina 413 è stato necessario aggiungere ulteriore codice sotto forma di routine di conversione finale. Dal momento che questa routine di conversione è piuttosto lunga, può risultare comodo non doverla riscrivere ogni volta, ma includerla nella libreria.
In questa appendice, utilizzeremo il gestore di libreria per includere la routine nella libreria IBM. Si tenga presente che sia la IBM che la Microsoft forniscono un gestore di libreria, ma solo la IBM fornisce la libreria IBMUTIL.LIB sul dischetto che contiene l'assembler.

D.2 Esempio di programma

Il programma che inizia a pagina 413 è abbastanza lungo e gran parte del codice viene impiegato per formattare i risultati piuttosto che per generarli. Le Figure D.1 e D.2 sono molto simili. La Figura D.1 mostra un programma che calcola la somma di due numeri reali, utilizzando il coprocessore 80287. La Figura D.2 mostra un programma che converte i risultati ottenuti precedentemente in un formato opportuno. Il programma presentato nel Capitolo 9 equivale ai due programmi di Figura D.1 e D.2. In questo esempio, viene indicato come salvare in libreria la parte di codice che esegue la formattazione dei risultati. Conclusa questa operazione, i programmi possono invocare la libreria di utilità quando i risultati elaborati dal coprocessore 80287 necessitano di essere formattati nella notazione scientifica.

```
COMMENT /Questo programma illustra l'uso del coprocessore 80287
per sommare due numeri reali. C'è una piccola parte di
istruzioni aggiuntive, necessarie per visualizzare le
risposte nel formato reale. La direttiva .8087 codifica
i numeri reali nel segmento dati per essere manipolati
dai coprocessori 8087 e 80287. Per visualizzare i
risultati, nel formato reale, il linker deve invocare
una routine di conversione (PRTREAL) della libreria
IBMUTIL. Questa routine converte un numero formattato
80287 in una stringa di caratteri e i risultati vengono
visualizzati sullo schermo in corrispondenza della
posizione corrente del cursore./

PAGE ,132
 .8087                                ;presenza coprocessore
COMMENT /I numeri reali possono essere nella forma:
                 123.4567         0.000048976
                 1.3E20          -4.5789E-3        /

         PUBLIC RIS                   ;variabile chiamata da libreria
DATA     SEGMENT PARA PUBLIC 'DATA'   ;deve essere Public
NUMERO1  DQ      1.2345E21            ;numero reale
NUMERO2  DQ     -13.456789E20         ;numero reale
RIS      DQ      ?
DATA     ENDS

CODICE   GROUP CODMAT                 ;nome richiesto del segmento di codice
CODMAT   SEGMENT BYTE PUBLIC 'CODE'   ;definisce il segmento di codice
PROCEDURA PROC   FAR                  ;inizio della procedura
         ASSUME CS:CODMAT,DS:DATA,ES:DATA
         EXTRN PRTREAL:NEAR           ;routine di libreria esterna
         PUSH   DS                    ;salva DS sullo stack
         SUB    AX,AX                 ;azzera AX
         PUSH   AX                    ;salva 0 sullo stack
         MOV    AX,DATI               ;indirizzo di DATI in AX
         MOV    DS,AX                 ;indirizzo di DATI in DS
         MOV    ES,AX                 ;indirizzo di DATI in ES
```

```
        LEA     SI,RIS                  ;l'indirizzo di RIS in SI
        FLD     NUMERO1                 ;carica NUMERO1 sullo stack dell'80287
        FADD    NUMERO2                 ;somma con NUMERO2
        FSTP    QWORD PTR [SI]          ;estrazione e memorizzazione del risultato
        FWAIT                           ;sincronizzazione con l'80286

        CALL    PRTREAL                 ;chiama PRTREAL per formattare il risultato

        RET                             ;il controllo ritorna al DOS
PROCEDURA ENDP                          ;fine della procedura
CODMAT  ENDS                            ;fine del segmento di codice

        END                             ;fine del programma
```

Figura D.1 Programma che calcola la somma di due numeri reali

```
         PUBLIC PRTREAL

DATA     SEGMENT PARA PUBLIC 'DATA'   ;il segmento dati deve essere Public
         EXTRN   RIS:QWORD            ;variabile da passare
BLANK    DB      20 DUP (' ')         ;stringa vuota per la routine IBM
RISPOSTA DB      17 DUP (?),'$'       ;risposta dalla routine IBM
PRIMA    DB      ' ','$'              ;locazione per la prima cifra
POT      DW      ?                    ;locazione per l'esponente
BUFF     DB      4 DUP (' ')          ;4 byte per i caratteri dell'esponente
SEGNO    DB      '-$'                 ;segno negativo
VIRGOLA  DB      '.$'                 ;virgola decimale
ESP      DB      ' E $'               ;simbolo dell'esponente
DATA     ENDS

CODICE   GROUP   SEGLIB
SEGLIB   SEGMENT BYTE PUBLIC 'CODE'
         ASSUME  CS:CODICE,DS:DATI
         EXTRN   $I8_OUTPUT:NEAR
PRTREAL  PROC    NEAR
         CALL    $I8_OUTPUT
         SUB     CX,01H               ;riduzione dell'esponente di 1
         MOV     POT,DX               ;DX contiene l'esponente corretto
         CMP     BL,'-'               ;verifica del segno della risposta
         JNE     NONEG
         LEA     DX,SEGNO             ;se il segno è negativo, - sullo schermo
         MOV     AH,09
         INT     21H
NONEG:
         CLD                          ;inizio trasferimento stringa in RISPOSTA
         MOV     CL,17                ;la stringa è di 17 byte
         LEA     DI,RISPOSTA[1]       ;destinazione del trasferimento
REP      MOVSB                        ;trasferimento
         MOV     CL,1                 ;primo carattere di RISPOSTA in PRIMA
         LEA     SI,RISPOSTA[1]
         LEA     DI,PRIMA
REP      MOVSB
         LEA     DX,PRIMA             ;visualizza la prima cifra sullo schermo
         MOV     AH,09
```

```
         INT     21H
         LEA     DX,VIRGOLA          ;visualizza la virgola decimale
         MOV     AH,09
         INT     21H
         LEA     DX,RISPOSTA[2]      ;visualizza le cifre restanti
         MOV     AH,09
         INT     21H
         LEA     DX,ESP              ;visualizza il carattere E (esponente)
         MOV     AH,09
         INT     21H
         MOV     DX,POT              ;conversione del numero in una stringa
         CMP     DX,8000H            ;è positivo o negativo?
         JB      POSIT               ;se è positivo, salta a POSIT
         LEA     DX,SEGNO            ;se è negativo, - sullo schermo
         MOV     AH,09
         INT     21H
         MOV     DX,POT              ;valore esadecimale corretto
         XOR     DX,0FFFFH
         ADD     DX,01
POSIT:   MOV     CX,0
         LEA     DI,BUFF             ;BUFF serve come memoria temporanea
POT1:    PUSH    CX                  ;per convertire i numeri esadecimali
         MOV     AX,DX               ;contenuti in DX nei valori decimali
         MOV     DX,0                ;ASCII da visualizzare sullo schermo
         MOV     CX,10
         DIV     CX
         XCHG    AX,DX
         ADD     AL,30H              ;conversione del numero in ASCII
         MOV     [DI],AL             ;memorizzazione del numero in BUFF
         INC     DI                  ;punta alla nuova locazione in BUFF
         POP     CX
         INC     CX                  ;CX contiene il numero di cifre
         CMP     DX,0
         JNZ     POT1
RIPETE:  DEC     DI                  ;preparativi per la visualizzazione
         MOV     AL,[DI]             ;cifra da BUFF a AL
         PUSH    DX                  ;salva il valore originale in DX
         MOV     DL,AL               ;trasferisce la cifra da visualizzare
         MOV     AH,2                ;parametro per visualizzazione DOS
         INT     21H                 ;visualizzazione
         POP     DX                  ;ripristino del valore originale di DX
         LOOP    RIPETE              ;continua fino alla fine
         RET
PRTREAL  ENDP                        ;fine della procedura PRTREAL
SEGLIB   ENDS                        ;fine del segmento di codice SEGLIB

         END                         ;fine del programma
```

Figura D.2 Programma che formatta l'uscita

Molti importanti dettagli che riguardano l'uso della libreria di utilità IBM sono stati discussi nel Capitolo 9 e sono contenuti nell'*IBM Macro Assembler Manual*, per cui è meglio controllare questo materiale prima di procedere oltre. Fate attenzione alle dichiarazioni Public per variabili, segmenti dati e segmenti di codice.

D.3 Uso del gestore di libreria

Il codice sorgente mostrato nelle Figure D.1 e D.2 deve essere tradotto dall'assembler che è memorizzato su dischetto. Dopo questa operazione, è possibile includere il codice di Figura D.2 nella libreria di utilità IBM.
Il gestore di libreria, anch'esso memorizzato su dischetto, permette all'utente di specificare cinque requisiti che devono essere definiti quando il gestore viene chiamato dal DOS: file di libreria, dimensione pagina, operazioni, file .LSF (catalogo) e nuova libreria. La sintassi di questo comando è:

LIB *file__lib*[*dim__pag*] [*operazioni*],[*file__list*],[*lib__nuova*]..[;]

Maggiori dettagli relativi ad ogni specifica sono contenuti nell'*IBM MASM Manual*. Nel nostro esempio, la libreria esiste già e si chiama IBMUTIL.LIB. Utilizziamo il gestore di libreria per determinare il contenuto di IBMUTIL.LIB.
Se l'assembler MASM IBM è contenuto nel drive C, scrivete da tastiera LIB, se il drive selezionato è C:

```
C>LIB

IBM Personal Computer Library Manager
Version 1.00
(C)Copyright IBM Corp 1984
(C)Copyright Microsoft Corp 1984

Library name: IBMUTIL
Operations:
List File: C:CONTENTS
```

In questa operazione, il gestore di libreria legge il contenuto di IBMUTIL, che si trova nel drive selezionato, e crea – sul dischetto contenuto nello stesso drive – un catalogo di nome CONTENTS (si tratta di un file ASCII che può essere visualizzato come qui di seguito riportato).

```
$I4_I8...........ifconv          $I4_M4...............bfconv
$I8_I4...........ifconv          $I8_INPUT............i8fin
$I8_M8...........bfconv          $I8_OUTPUT...........i8fout
$I8_TMUL.........i8tmul          $I8_TPWR10...........i8tmul
$M4_I4...........bfconv          $M8_I8...............bfconv


bfconv          Offset: 200H  Code and data size: F0
  $I4_M4          $I8_M8          $M4_I4          $M8_I8

ifconv          Offset: 400H  Code and data size: C0
  $I4_I8          $I8_I4

i8fin           Offset: 600H  Code and data size: 2FD
  $I8_INPUT
```

```
i8fout          Offset: C00H  Code and data size: 1A2
  $I8_OUTPUT

i8tmul          Offset: 1000H  Code and data size: 1EA
  $I8_TMUL        $I8_TPWR10
```

Il gestore di libreria può essere ora utilizzato per inserire il codice della Figura D.2 nella libreria di utilità IBM. Il nome del programma di Figura D.2 è PRTREAL. È possibile includere nella libreria solo file .OBJ, per cui è necessario prima eseguire la traduzione in codice oggetto.
Il comando per includere la routine viene riportato qui di seguito:

```
C>LIB IBMUTIL.LIB + PRTREAL.OBJ
```

È sufficiente scrivere da tastiera quest'unico comando. Analizzate il file .LSF per verificare se l'operazione è stata realizzata correttamente:

```
C>LIB


IBM Personal Computer Library Manager
Version 1.00
(C)Copyright IBM Corp 1984
(C)Copyright Microsoft Corp 1984

Library name: IBMUTIL
Operations:
List File: C:NEWINFO
```

Il file .LSF è stato memorizzato in NEWINFO sul drive C. Visualizzate il contenuto di NEWINFO sullo schermo:

```
$I4_I8...........ifconv          $I4_M4...............bfconv
$I8_I4...........ifconv          $I8_INPUT............i8fin
$I8_M8...........bfconv          $I8_OUTPUT...........i8fout
$I8_TMUL.........i8tmul          $I8_TPWR10...........i8tmul
$M4_I4...........bfconv          $M8_I8...............bfconv
PRTREAL..........PRTREAL


bfconv          Offset: 200H  Code and data size: F0
  $I4_M4          $I8_M8          $M4_I4          $M8_I8

ifconv          Offset: 400H  Code and data size: C0
  $I4_I8          $I8_I4

i8fin           Offset: 600H  Code and data size: 2FD
  $I8_INPUT

i8fout          Offset: C00H  Code and data size: 1A2
  $I8_OUTPUT

i8tmul          Offset: 1000H  Code and data size: 1EA
  $I8_TMUL        $I8_TPWR10
```

```
PRTREAL          Offest: 1400H  Code and data size: DB
  PRTREAL
```

Avendo incluso PRTREAL alla libreria di utilità IBM, è necessario specificare – in fase di collegamento – IBMUTIL come libreria:

```
C>LINK


IBM Personal Computer Linker
Version 2.30 (C) Copyright IBM Corp. 1981, 1985

Object Modules [.OBJ]: ADDER
Run File [ADDER.EXE]:
List File [NUL.MAP]
Libraries [.LIB]: IBMUTIL
```

Gran parte delle routine che abbiamo presentato in questo libro possono essere incluse alla vostra personale libreria, che potete creare con il gestore di libreria.

E

Codice ASCII dei caratteri

Tabella E.1

Dec.	Ott.	Esa.	ASCII	Dec.	Ott.	Esa.	ASCII
0	000	00	NUL	17	021	11	DC1
1	001	01	SOH	18	022	12	DC2
2	002	02	STX	19	023	13	DC3
3	003	03	ETX	20	024	14	DC4
4	004	04	EOT	21	025	15	NAK
5	005	05	ENQ	22	026	16	SYN
6	006	06	ACK	23	027	17	ETB
7	007	07	BEL	24	030	18	CAN
8	010	08	BS	25	031	19	EM
9	011	09	HT	26	032	1A	SUB
10	012	0A	LF	27	033	1B	ESC
11	013	0B	VT	28	034	1C	FS
12	014	0C	FF	29	035	1D	GS
13	015	0D	CR	30	036	1E	RS
14	016	0E	SO	31	037	1F	US
15	017	0F	SI	32	040	20	spazio
16	020	10	DLE	33	041	21	!

Tabella E.1 (continua)

Dec.	Ott.	Esa.	ASCII	Dec.	Ott.	Esa.	ASCII
34	042	22	”	76	114	4C	L
35	043	23	#	77	115	4D	M
36	044	24	$	78	116	4E	N
37	045	25	%	79	117	4F	O
38	046	26	&	80	120	50	P
39	047	27	’	81	121	51	Q
40	050	28	(	82	122	52	R
41	051	29	)	83	123	53	S
42	052	2A	*	84	124	54	T
43	053	2B	+	85	125	55	U
44	054	2C	,	86	126	56	V
45	055	2D	—	87	127	57	W
46	056	2E	.	88	130	58	X
47	057	2F	/	89	131	59	Y
48	060	30	0	90	132	5A	Z
49	061	31	1	91	133	5B	[
50	062	32	2	92	134	5C	\
51	063	33	3	93	135	5D	]
52	064	34	4	94	136	5E	^
53	065	35	5	95	137	5F	—
54	066	36	6	96	140	60	‘
55	067	37	7	97	141	61	a
56	070	38	8	98	142	62	b
57	071	39	9	99	143	63	c
58	072	3A	:	100	144	64	d
59	073	3B	;	101	145	65	e
60	074	3C	<	102	146	66	f
61	075	3D	=	103	147	67	g
62	076	3E	>	104	150	68	h
63	077	3F	?	105	151	69	i
64	100	40	@	106	152	6A	j
65	101	41	A	107	153	6B	k
66	102	42	B	108	154	6C	l
67	103	43	C	109	155	6D	m
68	104	44	D	110	156	6E	n
69	105	45	E	111	157	6F	o
70	106	46	F	112	160	70	p
71	107	47	G	113	161	71	q
72	110	48	H	114	162	72	r
73	111	49	I	115	163	73	s
74	112	4A	J	116	164	74	t
75	113	4B	K	117	165	75	u

Tabella E.1 (continua)

Dec.	Ott.	Esa.	ASCII	Dec.	Ott.	Esa.	ASCII
118	166	76	v	123	173	7B	{
119	167	77	w	124	174	7C	|
120	170	78	x	125	175	7D	}
121	171	79	y	126	176	7E	~
122	172	7A	z	127	177	7F	DEL

Indice analitico

Questo volume, sprovvisto del talloncino a fronte, è da considerarsi copia saggio e campione gratuito fuori commercio. Fuori campo applicazione IVA ed esente da bolla di accompagnamento (art. 22 L. 67/1987, art. 2 lett. i DPR 633/1972 e art. 4 n. 6 DPR 627/1978).

L. 64 000 (i.i.)

Murray - Pappas
L'assembler per l'80286/80386
McGraw-Hill Libri Italia
88 386 0049-X

I microprocessori Intel 80286/80386 occupano una consolidata posizione di mercato come CPU per i personal computer più avanzati; a conferma di tale leadership anche la nuova serie IBM Personal System/2 è sviluppata attorno a questi due microprocessori.
L'Assembler per l'80286/80386 è un testo che si prefigge principalmente tre scopi: introdurre alla programmazione in linguaggio assemblatore, insegnare le tecniche di programmazione sui processori 80286/80386 e fornire un testo di consultazione per conoscere e utilizzare correttamente le istruzioni del linguaggio e le soluzioni più adatte al tipo di problema da risolvere.
Tra le sezioni più approfondite troviamo:

- il set completo delle istruzioni
- le applicazioni dei coprocessori matematici 80287/80387 con il relativo set di istruzioni
- le direttive del linguaggio
- le macro, le procedure e le librerie
- le tecniche di debugging e di collaudo
- l'uso dei più diffusi assemblatori disponibili.

Tutte queste informazioni sono esemplificate con numerose routine applicative di complessità crescente che guidano il lettore fino alla completa padronanza del linguaggio.

Da un originale  Osborne/McGraw-Hill

ISBN 88-386-0049-X



Lire 64 000 (i.i.)

W.H. Murray
C.H. Pappas

L'Assembler per
l'80286/80386

049-X

